



《Spark编程基础（Python版）》

教材官网: <http://dbllab.xmu.edu.cn/post/spark-python/>

温馨提示: 编辑幻灯片母版, 可以修改每页PPT的厦大校徽和底部文字

第4章 RDD编程

(PPT版本号: 2020年1月版)



扫一扫访问教材官网

林子雨

厦门大学计算机科学系

E-mail: ziyulin@xmu.edu.cn ▶▶

主页: <http://dbllab.xmu.edu.cn/post/linziyu>



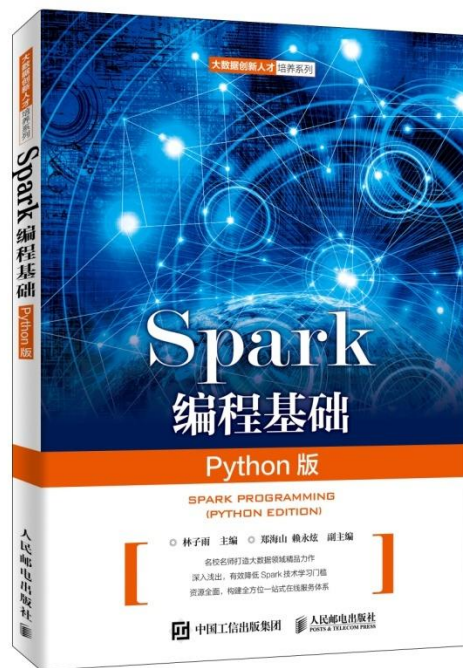


课程教材

林子雨，郑海山，赖永炫 编著 《Spark编程基础（Python版）》

教材官网：<http://dbllab.xmu.edu.cn/post/spark-python/>

ISBN:978-7-115-52439-3 人民邮电出版社



本书以Python作为开发Spark应用程序的编程语言，系统介绍了Spark编程的基础知识。全书共8章，内容包括大数据技术概述、Spark的设计与运行原理、Spark环境搭建和使用方法、RDD编程、Spark SQL、Spark Streaming、Structured Streaming、Spark MLlib等。本书每个章节都安排了入门级的编程实践操作，以便读者更好地学习和掌握Spark编程方法。本书官网免费提供了全套的在线教学资源，包括讲义PPT、习题、源代码、软件、数据集、授课视频、上机实验指南等。



提纲

4.1 RDD编程基础

4.2 键值对RDD

4.3 数据读写

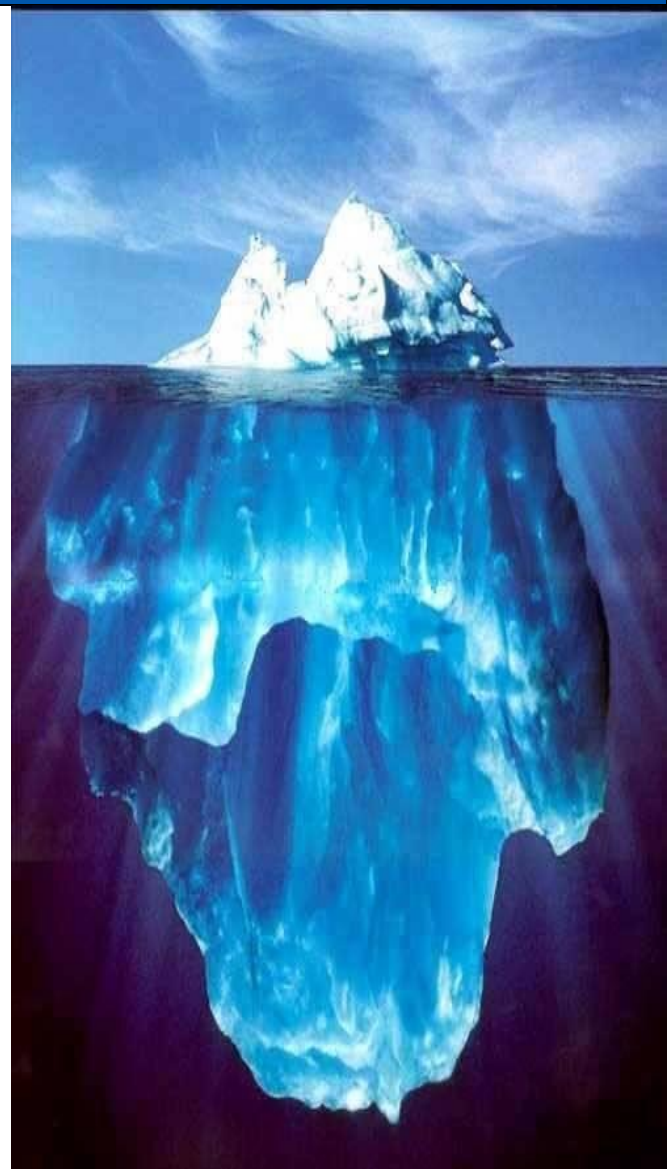
4.4 综合案例



高校大数据课程

公 共 服 务 平 台

百度搜索厦门大学数据库实验室网站访问平台





4.1 RDD编程基础

4.1.1 RDD创建

4.1.2 RDD操作

4.1.3 持久化

4.1.4 分区

4.1.5 一个综合实例



4.1.1 RDD创建

1. 从文件系统中加载数据创建RDD
2. 通过并行集合（列表）创建RDD



4.1.1 RDD创建

1. 从文件系统中加载数据创建RDD

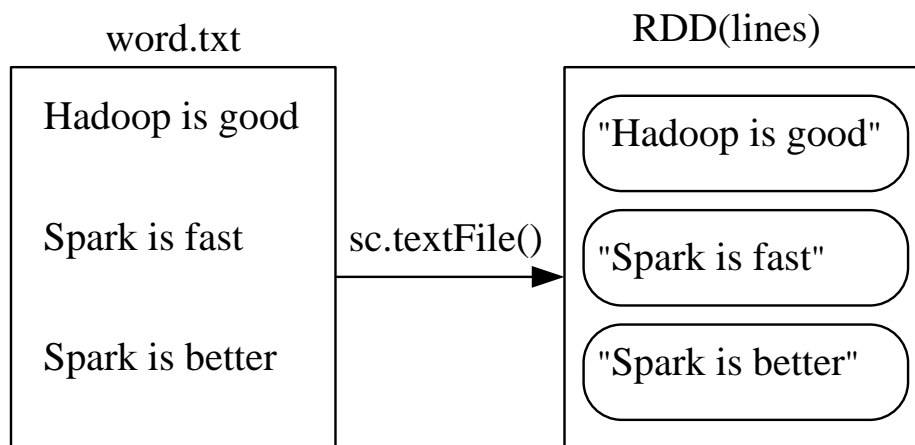
- Spark采用`textFile()`方法来从文件系统中加载数据创建RDD
- 该方法把文件的URI作为参数，这个URI可以是：
 - 本地文件系统的地址
 - 或者是分布式文件系统HDFS的地址
 - 或者是Amazon S3的地址等等



4.1.1 RDD创建

(1) 从本地文件系统中加载数据创建RDD

```
>>> lines = sc.textFile("file:///usr/local/spark/mycode/rdd/word.txt")
>>> lines.foreach(print)
Hadoop is good
Spark is fast
Spark is better
```



 表示一个RDD元素

图 从文件中加载数据生成RDD



4.1.1 RDD创建

(2) 从分布式文件系统HDFS中加载数据

```
>>> lines = sc.textFile("hdfs://localhost:9000/user/hadoop/word.txt")  
>>> lines = sc.textFile("/user/hadoop/word.txt")  
>>> lines = sc.textFile("word.txt")
```

三条语句是完全等价的，可以使用其中任意一种方式



4.1.1 RDD创建

2. 通过并行集合（列表）创建RDD

可以调用SparkContext的parallelize方法，在Driver中一个已经存在的集合（列表）上创建。

```
>>> array = [1,2,3,4,5]
>>> rdd = sc.parallelize(array)
>>> rdd.foreach(print)
1
2
3
4
5
```

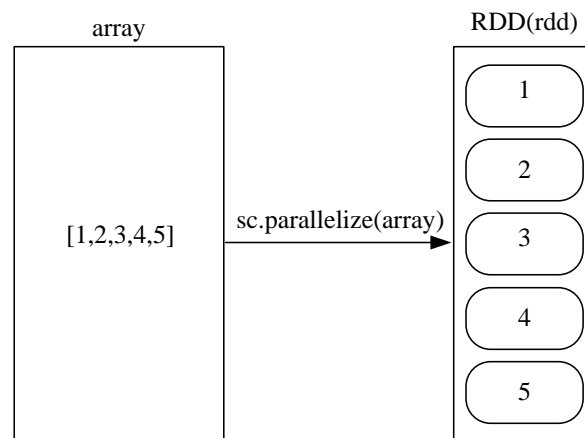


图 从数组创建RDD示意图



4.1.2 RDD操作

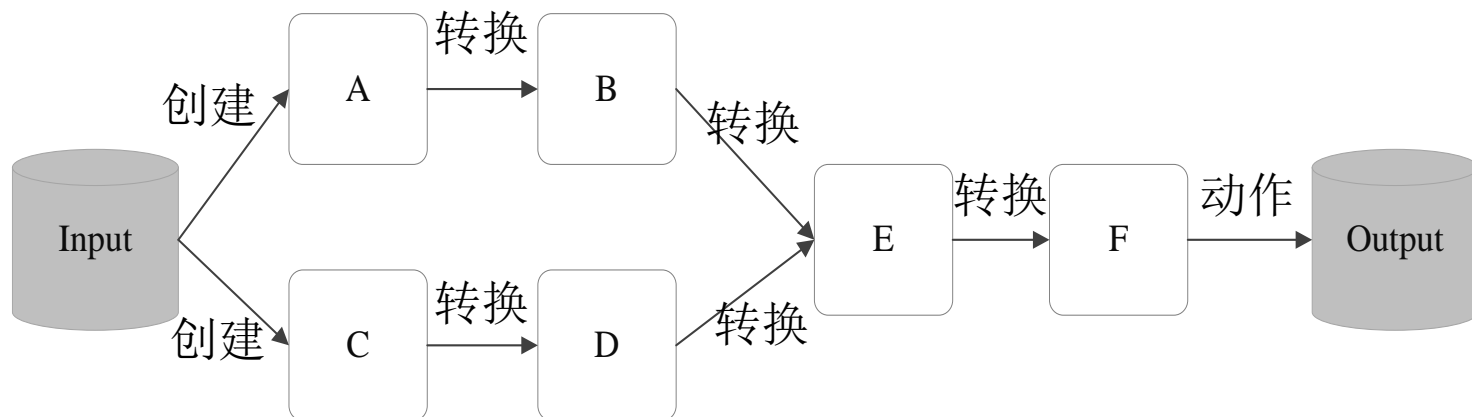
1. 转换操作
2. 行动操作
3. 惰性机制



4.1.2 RDD操作

1. 转换操作

- 对于**RDD**而言，每一次转换操作都会产生不同的**RDD**，供给下一个“转换”使用
- 转换得到的**RDD**是惰性求值的，也就是说，整个转换过程只是记录了转换的轨迹，并不会发生真正的计算，只有遇到行动操作时，才会发生真正的计算，开始从血缘关系源头开始，进行物理的转换操作





4.1.2 RDD操作

1. 转换操作

表 常用的RDD转换操作API

操作	含义
<code>filter(func)</code>	筛选出满足函数func的元素，并返回一个新的数据集
<code>map(func)</code>	将每个元素传递到函数func中，并将结果返回为一个新的数据集
<code>flatMap(func)</code>	与map()相似，但每个输入元素都可以映射到0或多个输出结果
<code>groupByKey()</code>	应用于(K,V)键值对的数据集时，返回一个新的(K, Iterable)形式的数据集
<code>reduceByKey(func)</code>	应用于(K,V)键值对的数据集时，返回一个新的(K, V)形式的数据集，其中每个值是将每个key传递到函数func中进行聚合后的结果



4.1.2 RDD操作

1. 转换操作

- `filter(func)`: 筛选出满足函数func的元素，并返回一个新的数据集

```
>>> lines = sc.textFile("file:///usr/local/spark/mycode/rdd/word.txt")
>>> linesWithSpark = lines.filter(lambda line: "Spark" in line)
>>> linesWithSpark.foreach(print)
Spark is better
Spark is fast
```

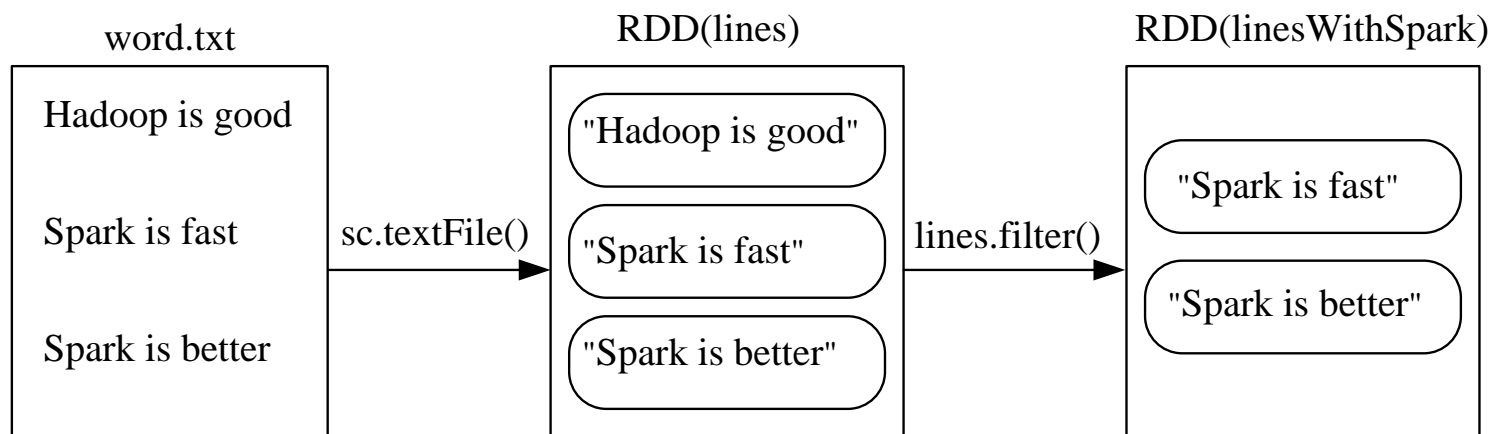


图 `filter()`操作实例执行过程示意图



4.1.2 RDD操作

1. 转换操作

- map(func)

map(func)操作将每个元素传递到函数func中，并将结果返回为一个新的数据集

```
>>> data = [1,2,3,4,5]
>>> rdd1 = sc.parallelize(data)
>>> rdd2 = rdd1.map(lambda x:x+10)
>>> rdd2.foreach(print)
11
13
12
14
15
```

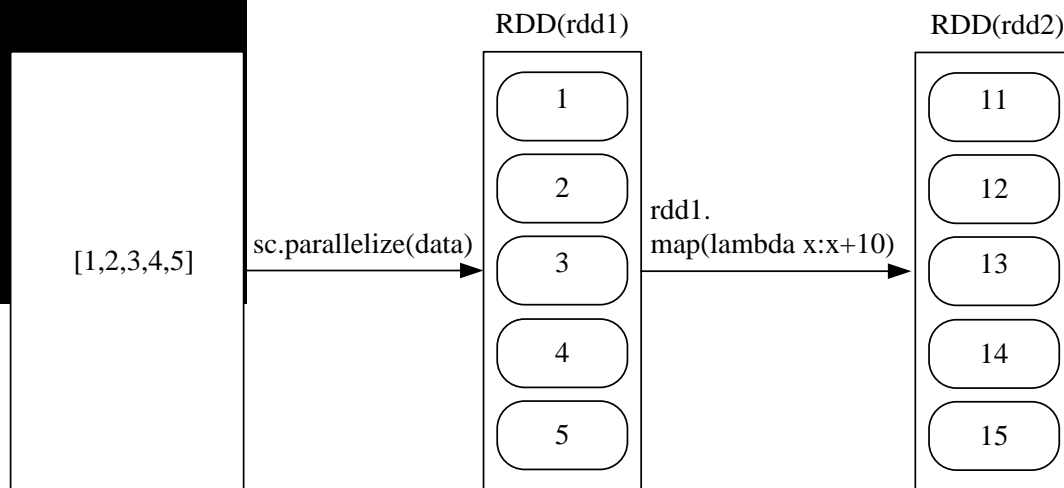


图 map()操作实例执行过程示意图



4.1.2 RDD操作

1. 转换操作

- map(func)

另外一个实例

```
>>> lines = sc.textFile("file:///usr/local/spark/mycode/rdd/word.txt")
>>> words = lines.map(lambda line:line.split(" "))
>>> words.foreach(print)
['Hadoop', 'is', 'good']
['Spark', 'is', 'fast']
['Spark', 'is', 'better']
```

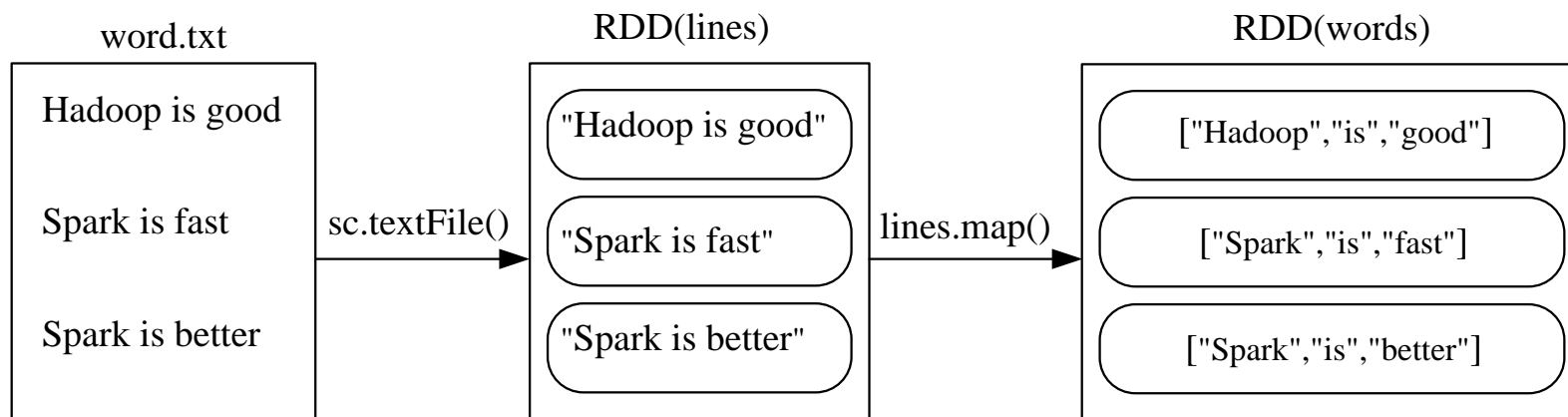


图 map()操作实例执行过程示意图



4.1.2 RDD操作

1. 转换操作

- flatMap(func)

```
>>> lines = sc.textFile("file:///usr/local/spark/mycode/rdd/word.txt")
>>> words = lines.flatMap(lambda line:line.split(" "))
```

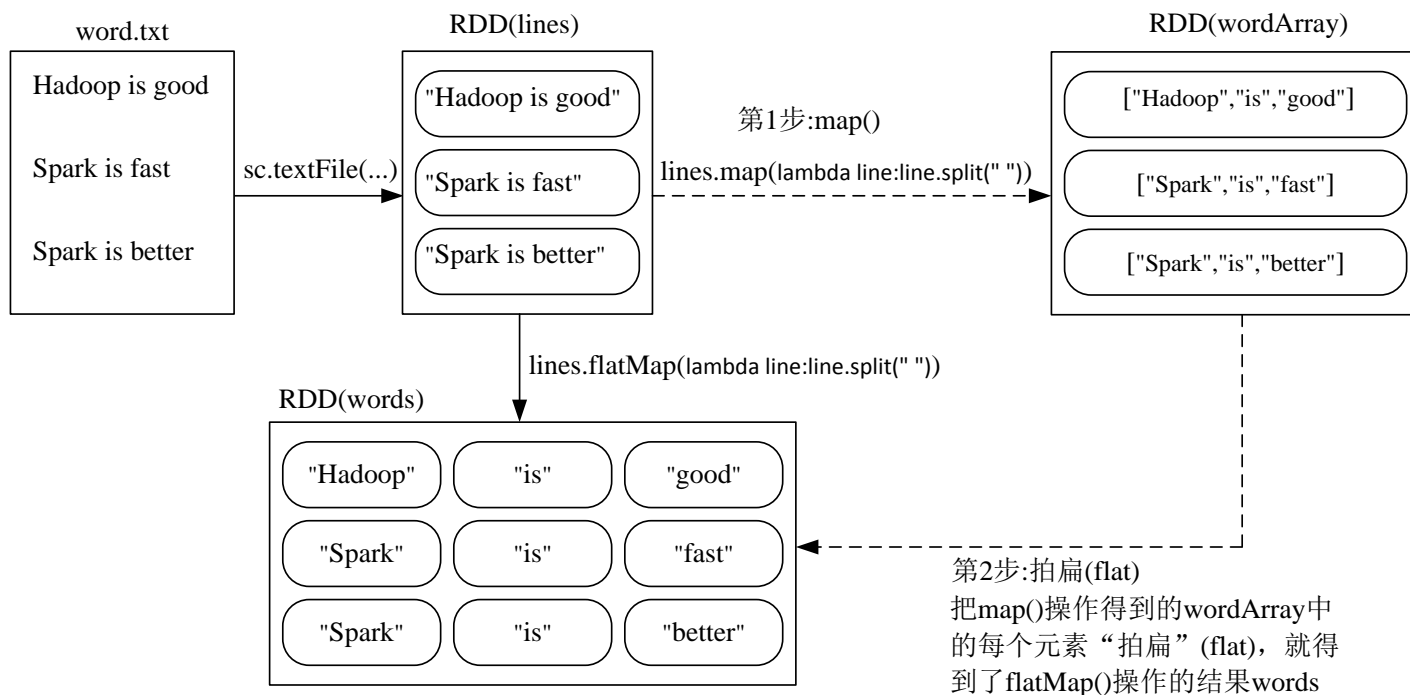


图 flatMap()操作实例执行过程示意图



4.1.2 RDD操作

1. 转换操作

- groupByKey()

groupByKey()应用于(K,V)键值对的数据集时，返回一个新的(K, Iterable)形式的数据集

```
>>> words = sc.parallelize([("Hadoop",1),("is",1),("good",1), \
... ("Spark",1),("is",1),("fast",1),("Spark",1),("is",1),("better",1)])
>>> words1 = words.groupByKey()
>>> words1.foreach(print)
('Hadoop', <pyspark.resultiterable.ResultIterable object at 0x7fb210552c88>)
('better', <pyspark.resultiterable.ResultIterable object at 0x7fb210552e80>)
('fast', <pyspark.resultiterable.ResultIterable object at 0x7fb210552c88>)
('good', <pyspark.resultiterable.ResultIterable object at 0x7fb210552c88>)
('Spark', <pyspark.resultiterable.ResultIterable object at 0x7fb210552f98>)
('is', <pyspark.resultiterable.ResultIterable object at 0x7fb210552e10>)
```

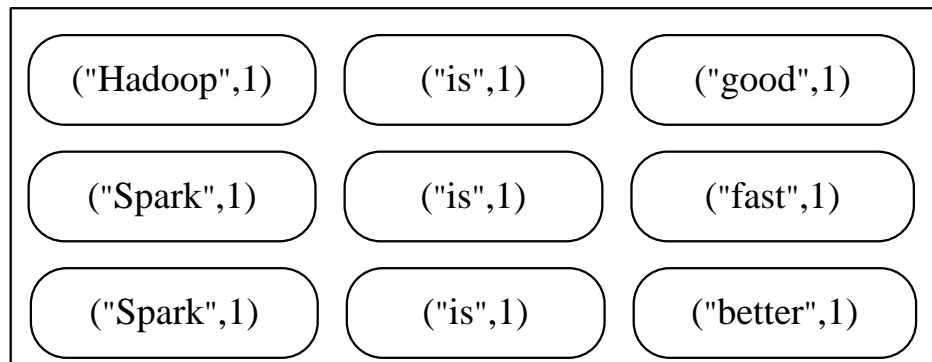


4.1.2 RDD操作

1. 转换操作

- groupByKey()

RDD(words)



words.groupByKey()

RDD(groupwords)

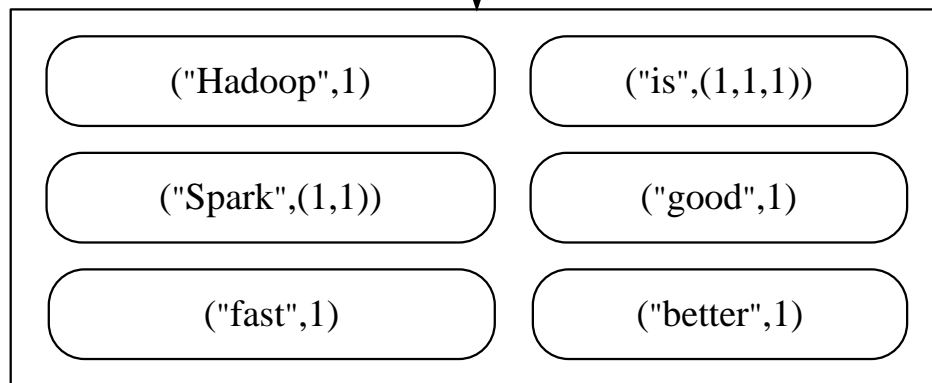


图 groupByKey()操作实例执行过程示意图



4.1.2 RDD操作

1. 转换操作

- reduceByKey(func)

reduceByKey(func)应用于(K,V)键值对的数据集时，返回一个新的(K, V)形式的数据集，其中的每个值是将每个key传递到函数func中进行聚合后得到的结果

```
>>> words = sc.parallelize([("Hadoop",1),("is",1),("good",1),("Spark",1), \
... ("is",1),("fast",1),("Spark",1),("is",1),("better",1)])
>>> words1 = words.reduceByKey(lambda a,b:a+b)
>>> words1.foreach(print)
('good', 1)
('Hadoop', 1)
('better', 1)
('Spark', 2)
('fast', 1)
('is', 3)
```



4.1.2 RDD操作

1. 转换操作

- `reduceByKey(func)`

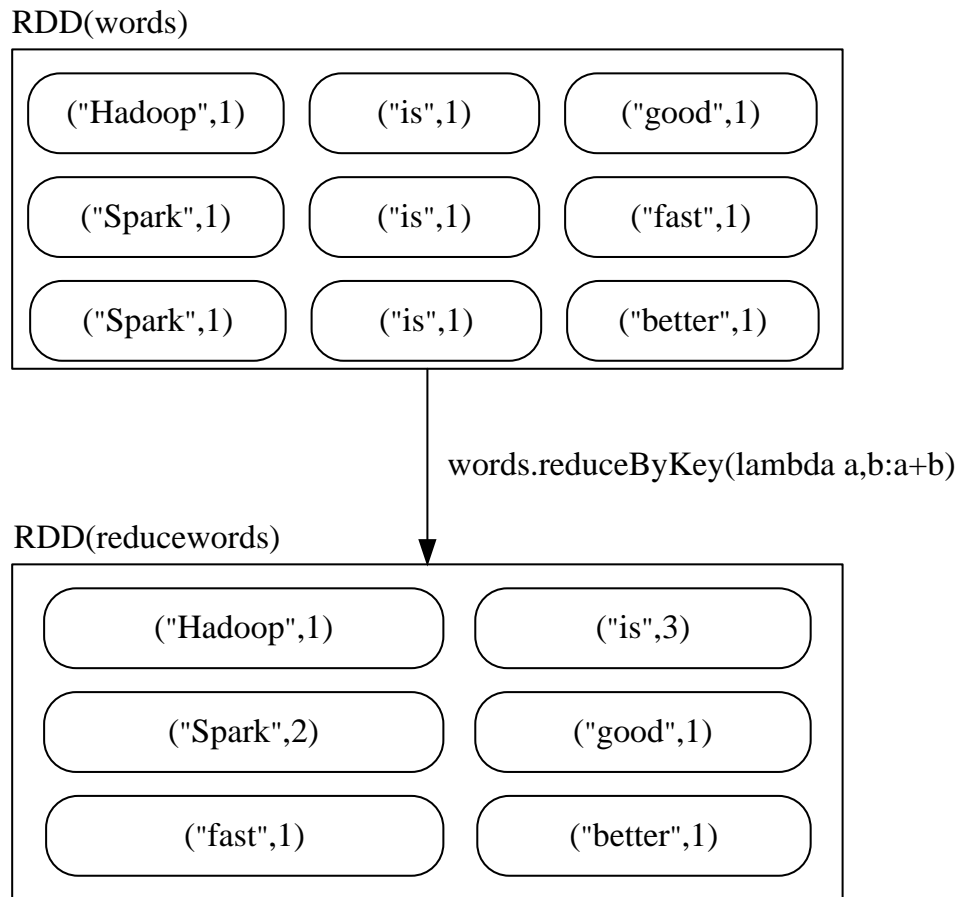


图 `reduceByKey()` 操作实例执行过程示意图

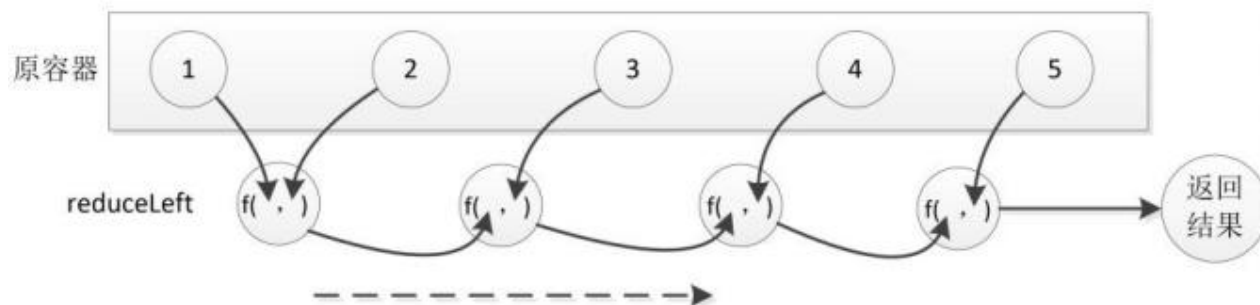


4.1.2 RDD操作

1. 转换操作

- `reduceByKey(func)`

`rdd.reduceByKey(lambda a,b:a+b)`



<“spark”,<1,1,1>>



4.1.2 RDD操作

2. 行动操作

行动操作是真正触发计算的地方。**Spark**程序执行到行动操作时，才会执行真正的计算，从文件中加载数据，完成一次又一次转换操作，最终，完成行动操作得到结果。

表 常用的RDD行动操作API

操作	含义
count()	返回数据集中的元素个数
collect()	以数组的形式返回数据集中的所有元素
first()	返回数据集中的第一个元素
take(n)	以数组的形式返回数据集中的前n个元素
reduce(func)	通过函数func（输入两个参数并返回一个值）聚合数据集中的元素
foreach(func)	将数据集中的每个元素传递到函数func中运行



4.1.2 RDD操作

```
>>> rdd = sc.parallelize([1,2,3,4,5])
>>> rdd.count()
5
>>> rdd.first()
1
>>> rdd.take(3)
[1, 2, 3]
>>> rdd.reduce(lambda a,b:a+b)
15
>>> rdd.collect()
[1, 2, 3, 4, 5]
>>> rdd.foreach(lambda elem:print(elem))
1
2
3
4
5
```



4.1.2 RDD操作

惰性机制

所谓的“惰性机制”是指，整个转换过程只是记录了转换的轨迹，并不会发生真正的计算，只有遇到行动操作时，才会触发“从头到尾”的真正的计算
这里给出一段简单的语句来解释Spark的惰性机制

```
>>> lines = sc.textFile("file:///usr/local/spark/mycode/rdd/word.txt")
>>> lineLengths = lines.map(lambda s:len(s))
>>> totalLength = lineLengths.reduce(lambda a,b:a+b)
>>> print(totalLength)
```




4.1.3 持久化

在Spark中，RDD采用惰性求值的机制，每次遇到行动操作，都会从头开始执行计算。每次调用行动操作，都会触发一次从头开始的计算。这对于迭代计算而言，代价是很大的，迭代计算经常需要多次重复使用同一组数据

下面就是多次计算同一个RDD的例子：

```
>>> list = ["Hadoop","Spark","Hive"]
>>> rdd = sc.parallelize(list)
>>> print(rdd.count()) //行动操作，触发一次真正从头到尾的计算
3
>>> print(','.join(rdd.collect())) //行动操作，触发一次真正从头到尾的计算
Hadoop,Spark,Hive
```



4.1.3 持久化

- 可以通过持久化（缓存）机制避免这种重复计算的开销
- 可以使用`persist()`方法对一个**RDD**标记为持久化
- 之所以说“标记为持久化”，是因为出现`persist()`语句的地方，并不会马上计算生成**RDD**并把它持久化，而是要等到遇到第一个行动操作触发真正计算以后，才会把计算结果进行持久化
- 持久化后的**RDD**将会被保留在计算节点的内存中被后面的行动操作重复使用



4.1.3 持久化

`persist()`的圆括号中包含的是持久化级别参数:

- `persist(MEMORY_ONLY)`: 表示将RDD作为反序列化的对象存储于JVM中, 如果内存不足, 就要按照LRU原则替换缓存中的内容
- `persist(MEMORY_AND_DISK)`表示将RDD作为反序列化的对象存储在JVM中, 如果内存不足, 超出的分区将会被存放在硬盘上
- 一般而言, 使用`cache()`方法时, 会调用`persist(MEMORY_ONLY)`
- 可以使用`unpersist()`方法手动地把持久化的RDD从缓存中移除



4.1.3 持久化

针对上面的实例，增加持久化语句以后的执行过程如下：

```
>>> list = ["Hadoop","Spark","Hive"]
>>> rdd = sc.parallelize(list)
>>> rdd.cache() #会调用persist(MEMORY_ONLY)，但是，语句执行到这里，
并不会缓存rdd，因为这时rdd还没有被计算生成
>>> print(rdd.count()) #第一次行动操作，触发一次真正从头到尾的计算，这
时上面的rdd.cache()才会被执行，把这个rdd放到缓存中
3
>>> print(', '.join(rdd.collect())) #第二次行动操作，不需要触发从头到尾的计算，
只需要重复使用上面缓存中的rdd
Hadoop,Spark,Hive
```



4.1.4 分区

RDD是弹性分布式数据集，通常**RDD**很大，会被分成很多个分区，分别保存在不同的节点上

1.分区的作用

(1) 增加并行度

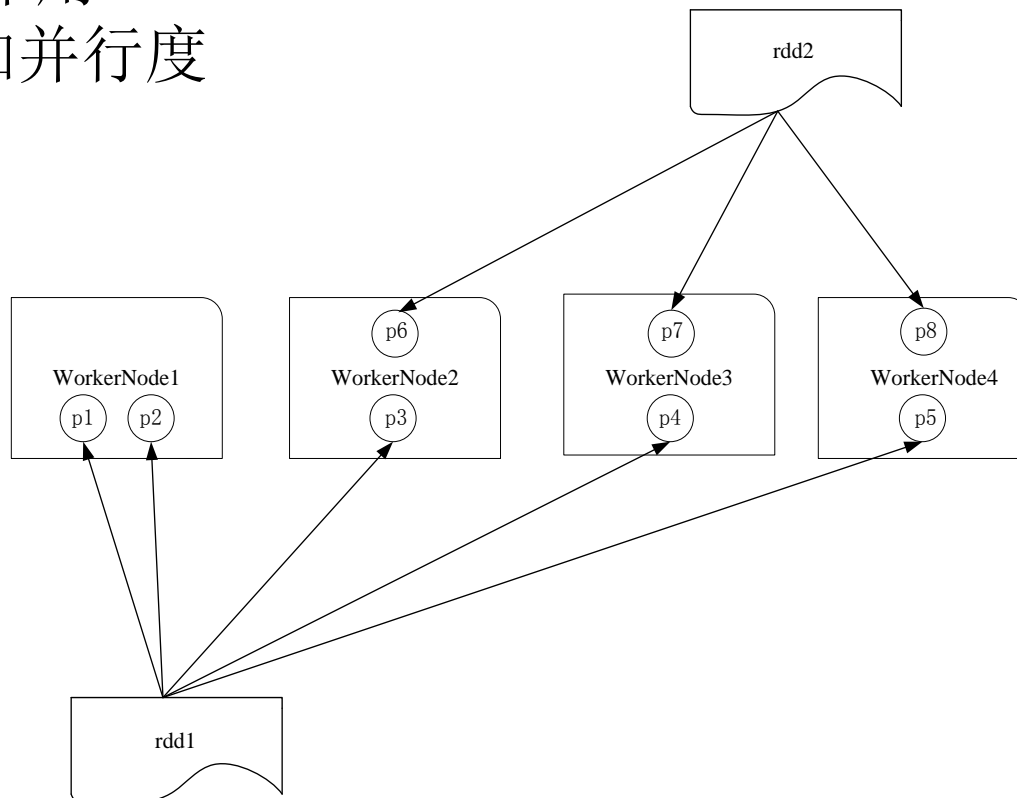


图 RDD分区被保存到不同节点上



4.1.4 分区

1.分区的作用

(2) 减少通信开销

UserData (UserId, UserInfo)

Events (UserID, LinkInfo)

UserData 和 Events 表进行连接操作, 获得
(UserID, UserInfo, LinkInfo)

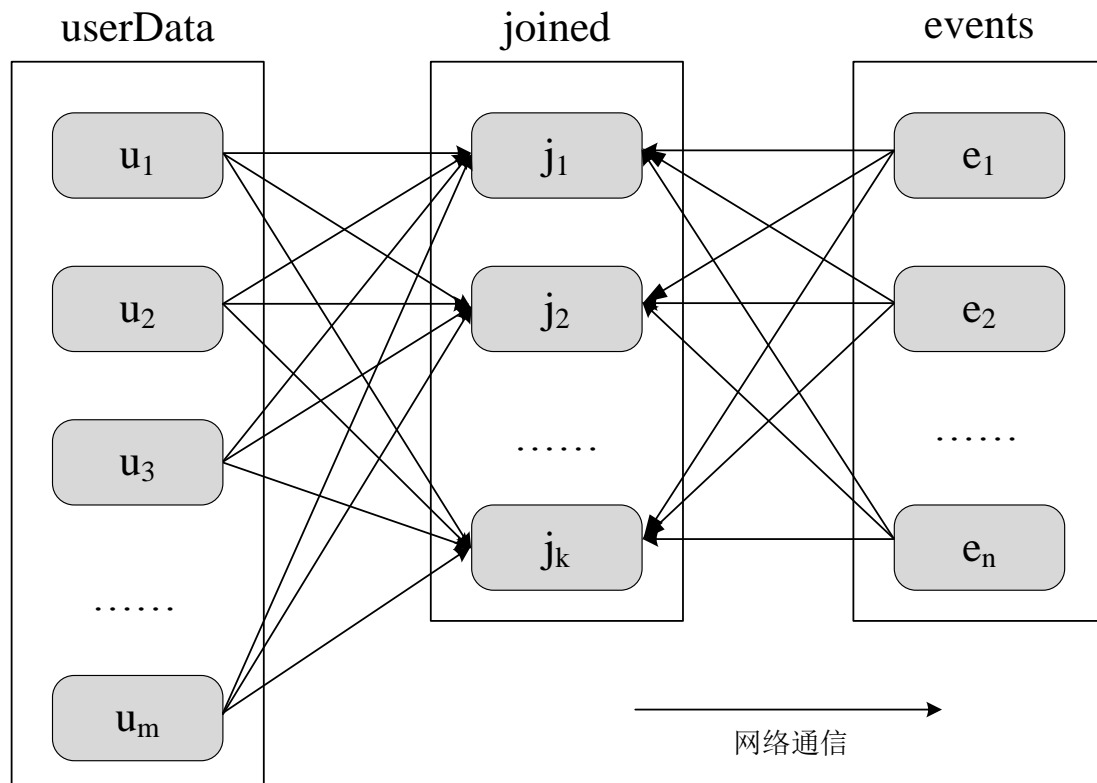


图 未分区时对UserData和Events两个表进行连接操作



4.1.4 分区

1.分区的作用

(2) 减少通信开销

UserData (UserId, UserInfo)

Events (UserID, LinkInfo)

UserData 和 Events 表进行连接操作, 获得
(UserID, UserInfo, LinkInfo)

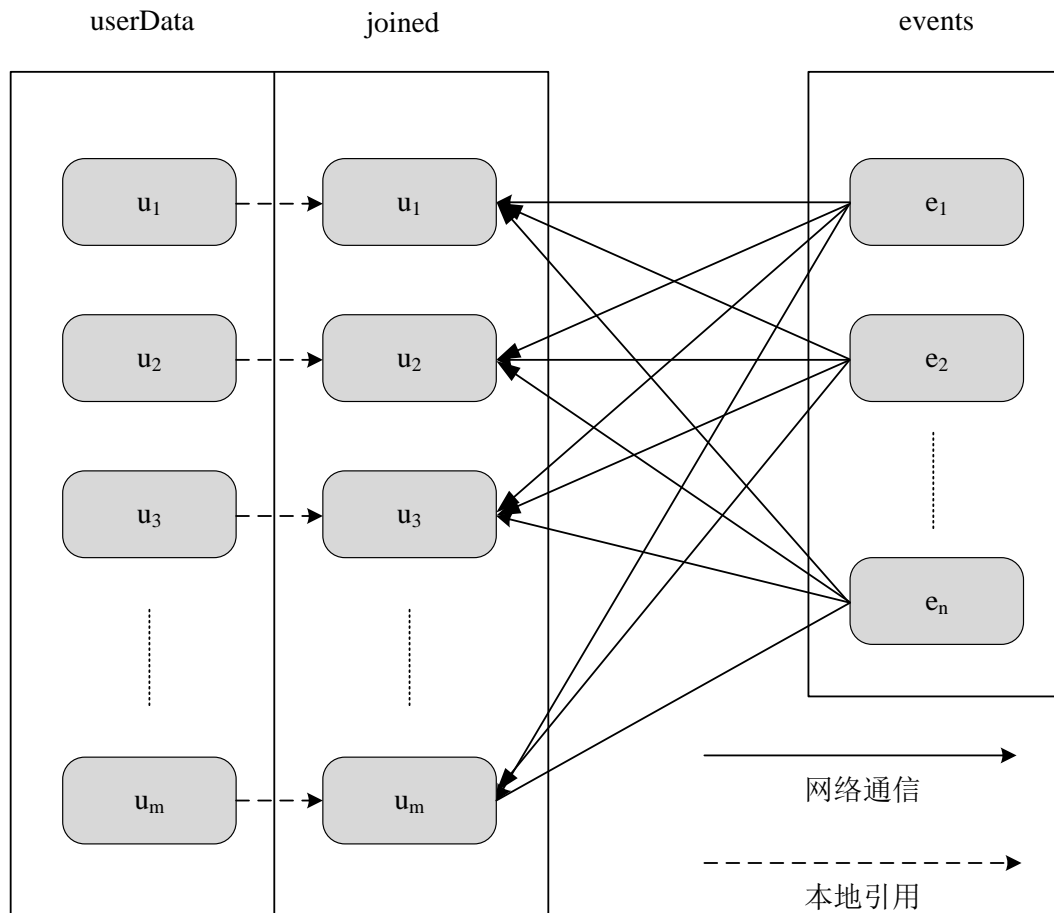


图 采用分区以后对UserData和Events两个表进行连接操作



4.1.4 分区

2.RDD分区原则

RDD分区的一个原则是使得分区的个数尽量等于集群中的CPU核心（core）数目

对于不同的Spark部署模式而言（本地模式、Standalone模式、YARN模式、Mesos模式），都可以通过设置 `spark.default.parallelism` 这个参数的值，来配置默认的分区数目，一般而言：

- *本地模式：默认为本地机器的CPU数目，若设置了 `local[N]`, 则默认为N

- *Apache Mesos：默认的分区数为8

- *Standalone或YARN：在“集群中所有CPU核心数目总和”和“2”二者中取较大值作为默认值



4.1.4 分区

3. 设置分区的个数

(1) 创建**RDD**时手动指定分区个数

在调用`textFile()`和`parallelize()`方法的时候手动指定分区个数即可，语法格式如下：

```
sc.textFile(path, partitionNum)
```

其中，`path`参数用于指定要加载的文件的地址，`partitionNum`参数用于指定分区个数。

```
>>> list = [1,2,3,4,5]
>>> rdd = sc.parallelize(list,2) //设置两个分区
```



4.1.4 分区

3.设置分区的个数

(2) 使用repartition方法重新设置分区个数

通过转换操作得到新 RDD 时，直接调用 repartition 方法即可。例如：

```
>>> data = sc.parallelize([1,2,3,4,5],2)
>>> len(data.glom().collect()) #显示data这个RDD的分区数量
2
>>> rdd = data.repartition(1) #对data这个RDD进行重新分区
>>> len(rdd.glom().collect()) #显示rdd这个RDD的分区数量
1
```



4.1.4 分区

4.自定义分区方法

Spark提供了自带的HashPartitioner（哈希分区）与RangePartitioner（区域分区），能够满足大多数应用场景的需求。与此同时，Spark也支持自定义分区方式，即通过提供一个自定义的分区函数来控制RDD的分区方式，从而利用领域知识进一步减少通信开销



4.1.4 分区

实例：根据**key**值的最后一位数字，写到不同的文件

例如：

10写入到part-00000

11写入到part-00001

.

.

.

19写入到part-00009



4.1.4 分区

TestPartitioner.py

```
from pyspark import SparkConf, SparkContext

def MyPartitioner(key):
    print("MyPartitioner is running")
    print('The key is %d' % key)
    return key%10

def main():
    print("The main function is running")
    conf = SparkConf().setMaster("local").setAppName("MyApp")
    sc = SparkContext(conf = conf)
    data = sc.parallelize(range(10),5)
    data.map(lambda x:(x,1)) \
        .partitionBy(10,MyPartitioner) \
        .map(lambda x:x[0]) \
        .saveAsTextFile("file:///usr/local/spark/mycode/rdd/partitioner")

if __name__ == '__main__':
    main()
```



4.1.4 分区

使用如下命令运行TestPartitioner.py:

```
$ cd /usr/local/spark/mycode/rdd  
$ python3 TestPartitioner.py
```

或者, 使用如下命令运行TestPartitioner.py:

```
$ cd /usr/local/spark/mycode/rdd  
$ /usr/local/spark/bin/spark-submit TestPartitioner.py
```

程序运行结果会返回如下信息:

```
The main function is running  
MyPartitioner is running  
The key is 0  
MyPartitioner is running  
The key is 1  
.....  
MyPartitioner is running  
The key is 9
```



4.1.5 一个综合实例

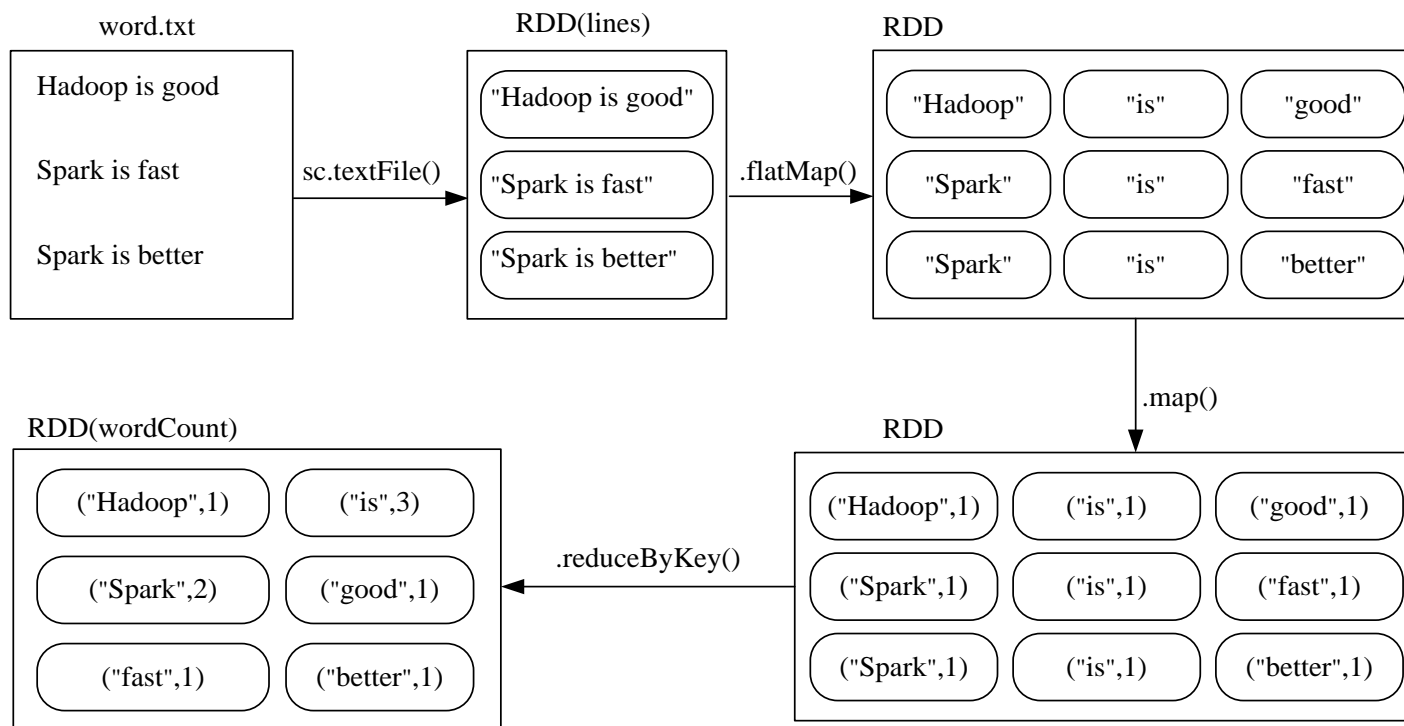
假设有一个本地文件`word.txt`，里面包含了很多行文本，每行文本由多个单词构成，单词之间用空格分隔。可以使用如下语句进行词频统计（即统计每个单词出现的次数）：

```
>>> lines = sc. \  
... textFile("file:///usr/local/spark/mycode/rdd/word.txt")  
>>> wordCount = lines.flatMap(lambda line:line.split(" ")). \  
... map(lambda word:(word,1)).reduceByKey(lambda a,b:a+b)  
>>> print(wordCount.collect())  
[('good', 1), ('Spark', 2), ('is', 3), ('better', 1), ('Hadoop', 1), ('fast', 1)]
```



4.1.5 一个综合实例

```
>>> lines = sc. \
...   textFile("file:///usr/local/spark/mycode/rdd/word.txt")
>>> wordCount = lines.flatMap(lambda line:line.split(" ")). \
...   map(lambda word:(word,1)).reduceByKey(lambda a,b:a+b)
>>> print(wordCount.collect())
[('good', 1), ('Spark', 2), ('is', 3), ('better', 1), ('Hadoop', 1), ('fast', 1)]
```





4.1.5 一个综合实例

在实际应用中，单词文件可能非常大，会被保存到分布式文件系统HDFS中，Spark和Hadoop会统一部署在一个集群上

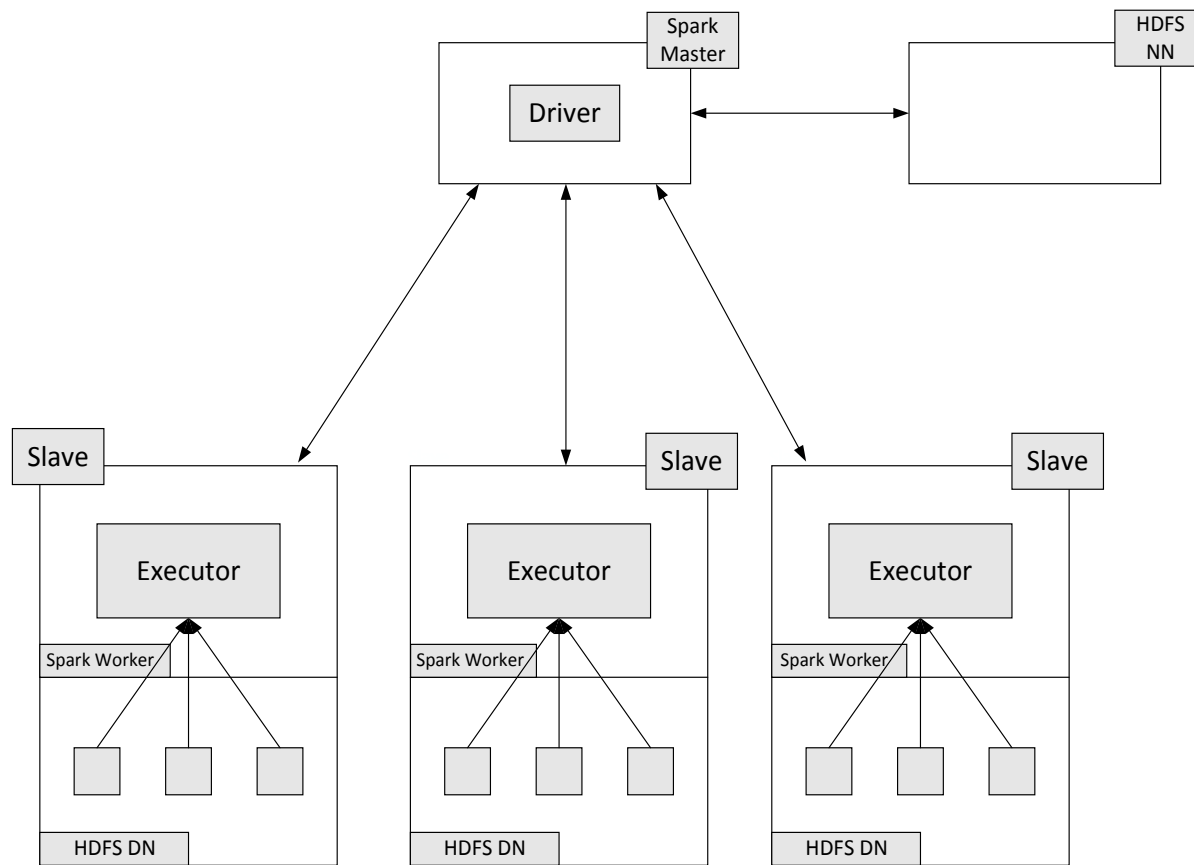


图 在一个集群中同时部署Hadoop和Spark



4.1.5 一个综合实例

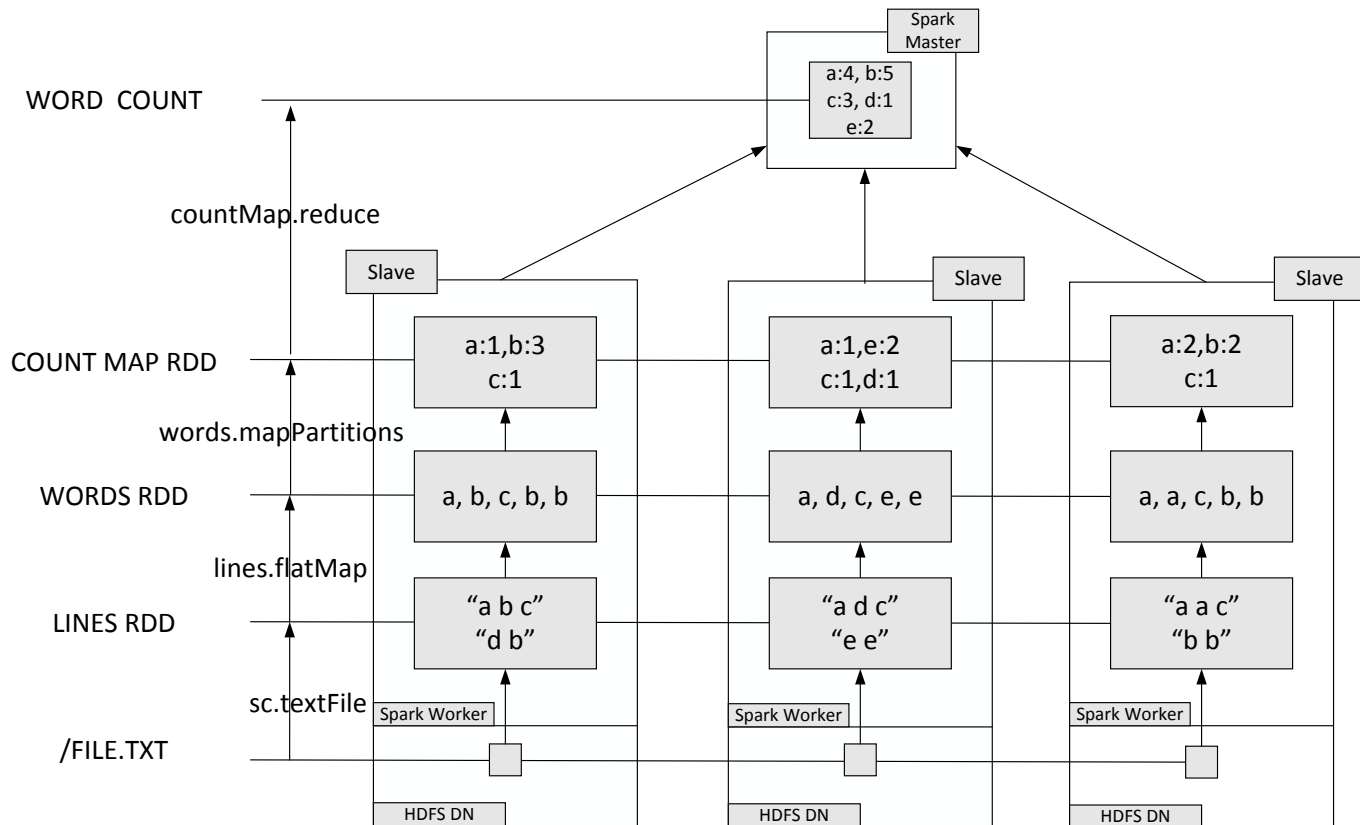


图 在集群中执行词频统计过程示意图



4.2 键值对RDD

4.2.1 键值对RDD的创建

4.2.2 常用的键值对RDD转换操作

4.2.3 一个综合实例



4.2.1 键值对RDD的创建

(1) 第一种创建方式：从文件中加载

可以采用多种方式创建键值对RDD，其中一种主要方式是使用map()函数来实现

```
>>> lines = sc.textFile("file:///usr/local/spark/mycode/pairrdd/word.txt")
>>> pairRDD = lines.flatMap(lambda line:line.split(" ")).map(lambda
word:(word,1))
>>> pairRDD.foreach(print)
('I', 1)
('love', 1)
('Hadoop', 1)
.....
```



4.2.1 键值对RDD的创建

(2) 第二种创建方式：通过并行集合（列表）创建RDD

```
>>> list = ["Hadoop","Spark","Hive","Spark"]
>>> rdd = sc.parallelize(list)
>>> pairRDD = rdd.map(lambda word:(word,1))
>>> pairRDD.foreach(print)
(Hadoop,1)
(Spark,1)
(Hive,1)
(Spark,1)
```



4.2.2 常用的键值对RDD转换操作

- **reduceByKey(func)**
- **groupByKey()**
- **keys**
- **values**
- **sortByKey()**
- **mapValues(func)**
- **join**
- **combineByKey**



4.2.2 常用的键值对RDD转换操作

•reduceByKey(func)

reduceByKey(func)的功能是，使用func函数合并具有相同键的值

(Hadoop,1)

(Spark,1)

(Hive,1)

(Spark,1)

```
>>> pairRDD = sc.parallelize([("Hadoop",1),("Spark",1),("Hive",1),("Spark",1)])
>>> pairRDD.reduceByKey(lambda a,b:a+b).foreach(print)
('Spark', 2)
('Hive', 1)
('Hadoop', 1)
```



4.2.2 常用的键值对RDD转换操作

•groupByKey()

groupByKey()的功能是，对具有相同键的值进行分组

比如，对四个键值对("spark",1)、("spark",2)、("hadoop",3)和("hadoop",5)，采用groupByKey()后得到的结果是：("spark",(1,2))和("hadoop",(3,5))

(Hadoop,1)

(Spark,1)

(Hive,1)

(Spark,1)

```
>>> list = [("spark",1),("spark",2),("hadoop",3),("hadoop",5)]
>>> pairRDD = sc.parallelize(list)
>>> pairRDD.groupByKey()
PythonRDD[27] at RDD at PythonRDD.scala:48
>>> pairRDD.groupByKey().foreach(print)
('hadoop', <pyspark.resultiterable.ResultIterable object at 0x7f2c1093ecf8>)
('spark', <pyspark.resultiterable.ResultIterable object at 0x7f2c1093ecf8>)
```




4.2.2 常用的键值对RDD转换操作

reduceByKey和**groupByKey**的区别

- **reduceByKey**用于对每个key对应的多个value进行merge操作，最重要的是它能够在本地先进行merge操作，并且merge操作可以通过函数自定义
- **groupByKey**也是对每个key进行操作，但只生成一个sequence，**groupByKey**本身不能自定义函数，需要先用**groupByKey**生成RDD，然后才能对此RDD通过map进行自定义函数操作



4.2.2 常用的键值对RDD转换操作

reduceByKey和groupByKey的区别

```
>>> words = ["one", "two", "two", "three", "three", "three"]
>>> wordPairsRDD = sc.parallelize(words).map(lambda word:(word, 1))
>>> wordCountsWithReduce = wordPairsRDD.reduceByKey(lambda a,b:a+b)
>>> wordCountsWithReduce.foreach(print)
('one', 1)
('two', 2)
('three', 3)
>>> wordCountsWithGroup = wordPairsRDD.groupByKey(). \
... map(lambda t:(t[0],sum(t[1])))
>>> wordCountsWithGroup.foreach(print)
('two', 2)
('three', 3)
('one', 1)
```

上面得到的wordCountsWithReduce和wordCountsWithGroup是完全一样的，但是，它们的内部运算过程是不同的



4.2.2 常用的键值对RDD转换操作

•keys

keys只会把Pair RDD中的key返回形成一个新的RDD

(Hadoop,1)

(Spark,1)

(Hive,1)

(Spark,1)

```
>>> list = [("Hadoop",1),("Spark",1),("Hive",1),("Spark",1)]
```

```
>>> pairRDD = sc.parallelize(list)
```

```
>>> pairRDD.keys().foreach(print)
```

```
Hadoop
```

```
Spark
```

```
Hive
```

```
Spark
```



4.2.2 常用的键值对RDD转换操作

- **values**

values只会把Pair RDD中的value返回形成一个新的RDD。

(Hadoop,1)

(Spark,1)

(Hive,1)

(Spark,1)

```
>>> list = [("Hadoop",1),("Spark",1),("Hive",1),("Spark",1)]
```

```
>>> pairRDD = sc.parallelize(list)
```

```
>>> pairRDD.values().foreach(print)
```

```
1
```

```
1
```

```
1
```

```
1
```



4.2.2 常用的键值对RDD转换操作

•sortByKey()

sortByKey()的功能是返回一个根据键排序的RDD

(Hadoop,1) (Spark,1) (Hive,1) (Spark,1)

```
>>> list = [("Hadoop",1),("Spark",1),("Hive",1),("Spark",1)]
>>> pairRDD = sc.parallelize(list)
>>> pairRDD.foreach(print)
('Hadoop', 1)
('Spark', 1)
('Hive', 1)
('Spark', 1)
>>> pairRDD.sortByKey().foreach(print)
('Hadoop', 1)
('Hive', 1)
('Spark', 1)
('Spark', 1)
```



4.2.2 常用的键值对RDD转换操作

•sortByKey()和sortBy()

```
>>> d1 = sc.parallelize([("c",8),("b",25),("c",17),("a",42), \
... ("b",4),("d",9),("e",17),("c",2),("f",29),("g",21),("b",9)])
>>> d1.reduceByKey(lambda a,b:a+b).sortByKey(False).collect()
[('g', 21), ('f', 29), ('e', 17), ('d', 9), ('c', 27), ('b', 38), ('a', 42)]
```

```
>>> d1 = sc.parallelize([("c",8),("b",25),("c",17),("a",42), \
... ("b",4),("d",9),("e",17),("c",2),("f",29),("g",21),("b",9)])
>>> d1.reduceByKey(lambda a,b:a+b).sortBy(lambda x:x,False).collect()
[('g', 21), ('f', 29), ('e', 17), ('d', 9), ('c', 27), ('b', 38), ('a', 42)]
>>> d1.reduceByKey(lambda a,b:a+b).sortBy(lambda x:x[0],False).collect()
[('g', 21), ('f', 29), ('e', 17), ('d', 9), ('c', 27), ('b', 38), ('a', 42)]
>>> d1.reduceByKey(lambda a,b:a+b).sortBy(lambda x:x[1],False).collect()
[('a', 42), ('b', 38), ('f', 29), ('c', 27), ('g', 21), ('e', 17), ('d', 9)]
```



4.2.2 常用的键值对RDD转换操作

•mapValues(func)

对键值对RDD中的每个value都应用一个函数，但是，key不会发生变化

(Hadoop,1)

(Spark,1)

(Hive,1)

(Spark,1)

```
>>> list = [("Hadoop",1),("Spark",1),("Hive",1),("Spark",1)]
>>> pairRDD = sc.parallelize(list)
>>> pairRDD1 = pairRDD.mapValues(lambda x:x+1)
>>> pairRDD1.foreach(print)
('Hadoop', 2)
('Spark', 2)
('Hive', 2)
('Spark', 2)
```



4.2.2 常用的键值对RDD转换操作

•join

join就表示内连接。对于内连接，对于给定的两个输入数据集(K,V1)和(K,V2)，只有在两个数据集中都存在的key才会被输出，最终得到一个(K,(V1,V2))类型的数据集。

```
>>> pairRDD1 = sc. \  
... parallelize([("spark",1),("spark",2),("hadoop",3),("hadoop",5)])  
>>> pairRDD2 = sc.parallelize([("spark","fast")])  
>>> pairRDD3 = pairRDD1.join(pairRDD2)  
>>> pairRDD3.foreach(print)  
(('spark', (1, 'fast'))  
(('spark', (2, 'fast'))
```

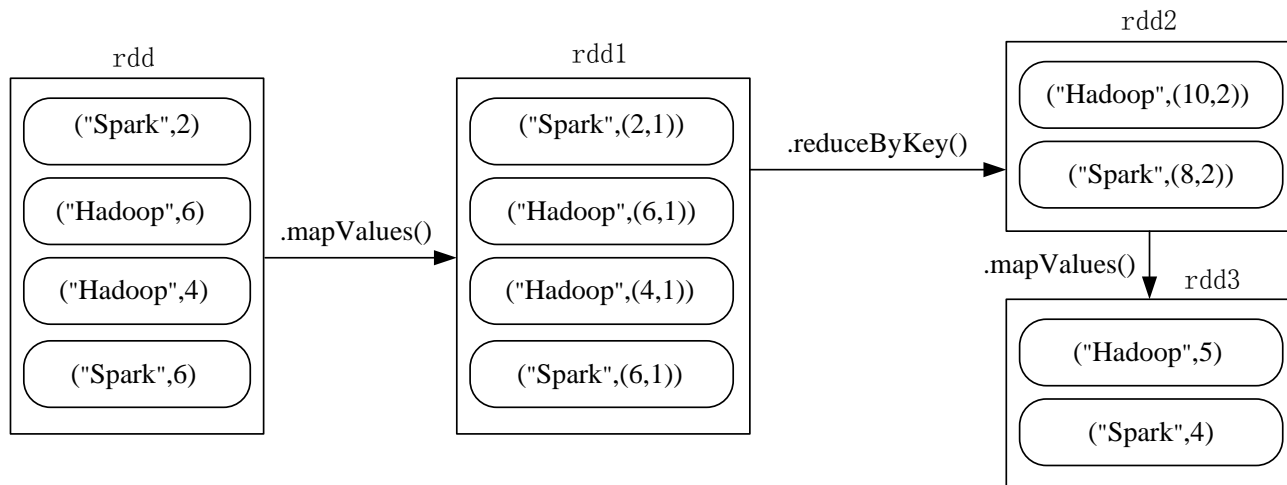



4.2.3 一个综合实例

一个综合实例

题目：给定一组键值对("spark",2),("hadoop",6),("hadoop",4),("spark",6)，键值对的key表示图书名称，value表示某天图书销量，请计算每个键对应的平均值，也就是计算每种图书的每天平均销量。

```
>>> rdd = sc.parallelize([("spark",2),("hadoop",6),("hadoop",4),("spark",6)])
>>> rdd.mapValues(lambda x:(x,1)).\
... reduceByKey(lambda x,y:(x[0]+y[0],x[1]+y[1])).\
... mapValues(lambda x:x[0]/x[1]).collect()
[('hadoop', 5.0), ('spark', 4.0)]
```





4.3 数据读写

4.3.1 文件数据读写

4.3.2 读写HBase数据



4.3.1 文件数据读写

1. 本地文件系统的数据读写
2. 分布式文件系统HDFS的数据读写



4.3.1 文件数据读写

1.本地文件系统的数据读写

(1) 从文件中读取数据创建RDD

```
>>> textFile = sc.\n... textFile("file:///usr/local/spark/mycode/rdd/word.txt")\n>>> textFile.first()\n'Hadoop is good'
```

因为Spark采用了惰性机制，在执行转换操作的时候，即使输入了错误的语句，spark-shell也不会马上报错（假设word123.txt不存在）

```
>>> textFile = sc.\n... textFile("file:///usr/local/spark/mycode/wordcount/word123.txt")
```



4.3.1 文件数据读写

1.本地文件系统的数据读写

(2) 把RDD写入到文本文件中

```
>>> textFile = sc.\n... textFile("file:///usr/local/spark/mycode/rdd/word.txt")\n>>> textFile.\n... saveAsTextFile("file:///usr/local/spark/mycode/rdd/writeback")
```

```
$ cd /usr/local/spark/mycode/wordcount/writeback/\n$ ls
```

```
part-00000\n_SUCCESS
```

如果想再次把数据加载在RDD中，只要使用writeback这个目录即可，如下：

```
>>> textFile = sc.\n... textFile("file:///usr/local/spark/mycode/rdd/writeback")
```



4.3.1 文件数据读写

2. 分布式文件系统HDFS的数据读写

从分布式文件系统HDFS中读取数据，也是采用`textFile()`方法，可以为`textFile()`方法提供一个HDFS文件或目录地址，如果是一个文件地址，它会加载该文件，如果是一个目录地址，它会加载该目录下的所有文件的数据

```
>>> textFile = sc.textFile("hdfs://localhost:9000/user/hadoop/word.txt")
>>> textFile.first()
```

如下三条语句都是等价的：

```
>>> textFile = sc.textFile("hdfs://localhost:9000/user/hadoop/word.txt")
>>> textFile = sc.textFile("/user/hadoop/word.txt")
>>> textFile = sc.textFile("word.txt")
```

同样，可以使用`saveAsTextFile()`方法把RDD中的数据保存到HDFS文件中，命令如下：

```
>>> textFile = sc.textFile("word.txt")
>>> textFile.saveAsTextFile("writeback")
```



4.3.2 读写HBase数据

- 0. HBase简介
- 1. 创建一个HBase表
- 2. 配置Spark
- 3. 编写程序读取HBase数据
- 4. 编写程序向HBase写入数据



4.3.2 读写HBase数据

0. HBase简介

- HBase是Google BigTable的开源实现

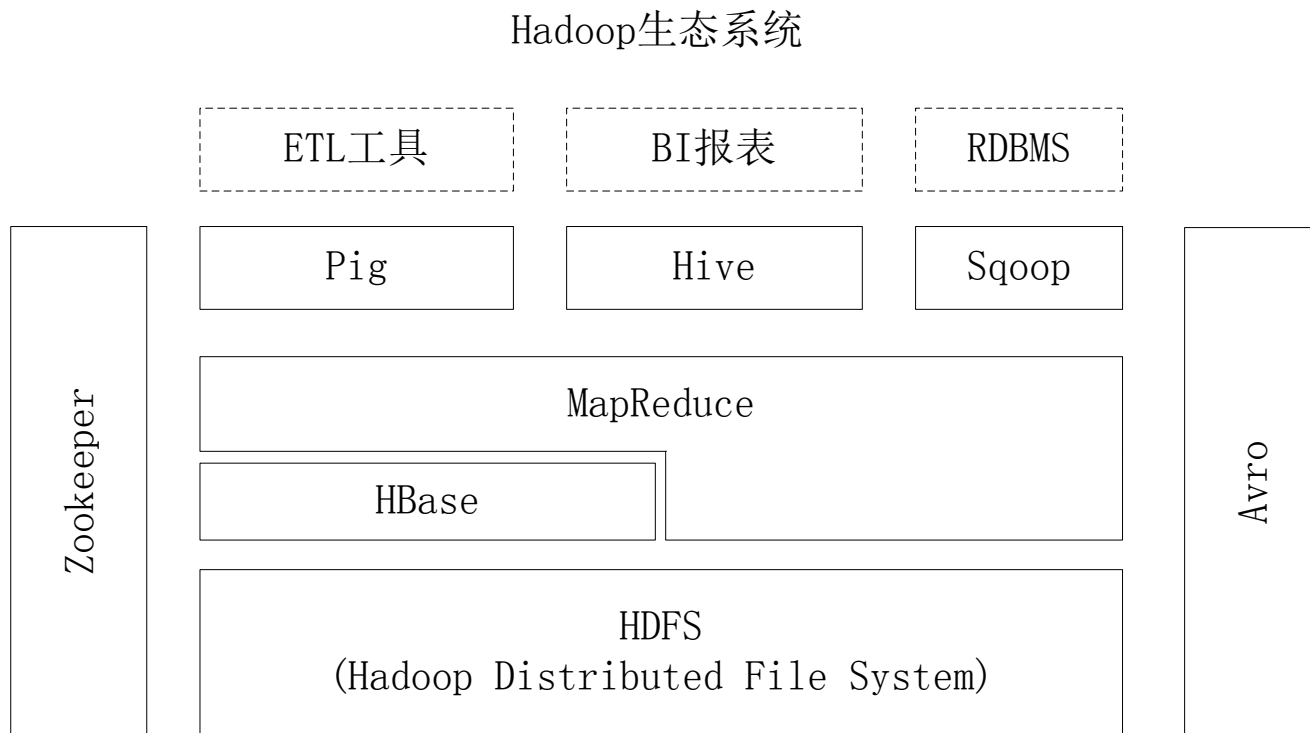


图 Hadoop生态系统中HBase与其他部分的关系



4.3.2 读写HBase数据

- **HBase**是一个稀疏、多维度、排序的映射表，这张表的索引是行键、列族、列限定符和时间戳
- 每个值是一个未经解释的字符串，没有数据类型
- 用户在表中存储数据，每一行都有一个可排序的行键和任意多的列
- 表在水平方向由一个或者多个列族组成，一个列族中可以包含任意多个列，同一个列族里面的数据存储在—起
- 列族支持动态扩展，可以很轻松地添加一个列族或列，无需预先定义列的数量以及类型，所有列均以字符串形式存储，用户需要自行进行数据类型转换
- **HBase**中执行更新操作时，并不会删除数据旧的版本，而是生成一个新的版本，旧有的版本仍然保留（这是和**HDFS**只允许追加不允许修改的特性相关的）



4.3.2 读写HBase数据

- 表: **HBase**采用表来组织数据, 表由行和列组成, 列划分为若干个列族
- 行: 每个**HBase**表都由若干行组成, 每个行由行键 (**row key**) 来标识。
- 列族: 一个**HBase**表被分组成许多“列族” (**Column Family**) 的集合, 它是基本的访问控制单元
- 列限定符: 列族里的数据通过列限定符 (或列) 来定位
- 单元格: 在**HBase**表中, 通过行、列族和列限定符确定一个“单元格” (**cell**), 单元格中存储的数据没有数据类型, 总被视为字节数组**byte[]**
- 时间戳: 每个单元格都保存着同一份数据的多个版本, 这些版本采用时间戳进行索引

	Info		
	name	major	email
201505001	Luo Min	Math	luo@qq.com
201505002	Liu Jun	Math	liu@qq.com
201505003	Xie You	Math	xie@qq.com you@163.com

该单元格有2个时间戳ts1和ts2
每个时间戳对应一个数据版本
ts1=1174184619081 ts2=1174184620720



4.3.2 读写HBase数据

- HBase中需要根据行键、列族、列限定符和时间戳来确定一个单元格，因此，可以视为一个“四维坐标”，即[行键, 列族, 列限定符, 时间戳]

键	值
["201505003", "Info", "email", 1174184619081]	"xie@qq.com"
["201505003", "Info", "email", 1174184620720]	"you@163.com"





4.3.2 读写HBase数据

表 HBase数据的概念视图

行键	时间戳	列族contents	列族anchor
"com.cnn .www"	t5		anchor:cnnsi.com="CNN"
	t4		anchor:my.look.ca="CNN.com"
	t3	contents:html="<html>..."	
	t2	contents:html="<html>..."	
	t1	contents:html="<html>..."	



4.3.2 读写HBase数据

表 HBase数据的物理视图
列族contents

行键	时间戳	列族contents
"com.cnn.www"	t3	contents:html="<html>..."
	t2	contents:html="<html>..."
	t1	contents:html="<html>..."

列族anchor

行键	时间戳	列族anchor
"com.cnn.www"	t5	anchor:cnnsi.com="CNN"
	t4	anchor:my.look.ca="CNN.com"



4.3.2 读写HBase数据

1. 创建一个HBase表

首先，请参照厦门大学数据库实验室博客完成HBase的安装（伪分布式模式）：

<http://dblab.xmu.edu.cn/blog/install-hbase/>

因为HBase是伪分布式模式，需要调用HDFS，所以，请首先在终端中输入下面命令启动Hadoop：

```
$ cd /usr/local/hadoop
$ ./sbin/start-all.sh
```

下面就可以启动HBase，命令如下：

```
$ cd /usr/local/hbase
$ ./bin/start-hbase.sh //启动HBase
$ ./bin/hbase shell //启动hbase shell
```

如果里面已经有一个名称为student的表，请使用如下命令删除：

```
hbase> disable 'student'
hbase> drop 'student'
```



4.3.2 读写HBase数据

1. 创建一个HBase表

下面创建一个student表，要在这个表中录入如下数据：

id	name	gender	age
1	Xueqian	F	23
2	Weiliang	M	24

```
hbase> create 'student','info'
```

```
// 首先录入student表的第一个学生记录
hbase> put 'student','1','info:name','Xueqian'
hbase> put 'student','1','info:gender','F'
hbase> put 'student','1','info:age','23'
// 然后录入student表的第二个学生记录
hbase> put 'student','2','info:name','Weiliang'
hbase> put 'student','2','info:gender','M'
hbase> put 'student','2','info:age','24'
```



4.3.2 读写HBase数据

2. 配置Spark

把HBase的lib目录下的一些jar文件拷贝到Spark中，这些都是编程时需要引入的jar包，需要拷贝的jar文件包括：所有hbase开头的jar文件、guava-12.0.1.jar、htrace-core-3.1.0-incubating.jar和protobuf-java-2.5.0.jar

执行如下命令：

```
$ cd /usr/local/spark/jars
$ mkdir hbase
$ cd hbase
$ cp /usr/local/hbase/lib/hbase*.jar ./
$ cp /usr/local/hbase/lib/guava-12.0.1.jar ./
$ cp /usr/local/hbase/lib/htrace-core-3.1.0-incubating.jar ./
$ cp /usr/local/hbase/lib/protobuf-java-2.5.0.jar ./
```




4.3.2 读写HBase数据

此外，在Spark 2.0以上版本中，缺少把HBase数据转换成Python可读取数据的jar包，需要另行下载。可以访问下面地址下载spark-examples_2.11-1.6.0-typesafe-001.jar:

https://mvnrepository.com/artifact/org.apache.spark/spark-examples_2.11/1.6.0-typesafe-001

下载以后保存到 “/usr/local/spark/jars/hbase/” 目录中



4.3.2 读写HBase数据

然后，使用vim编辑器打开spark-env.sh文件，设置Spark的spark-env.sh文件，告诉Spark可以在哪个路径下找到HBase相关的jar文件，命令如下：

```
$ cd /usr/local/spark/conf  
$ vim spark-env.sh
```

打开spark-env.sh文件以后，可以在文件最前面增加下面一行内容：

```
export SPARK_DIST_CLASSPATH=$(/usr/local/hadoop/bin/hadoop  
classpath):$(/usr/local/hbase/bin/hbase classpath):/usr/local/spark/jars/hbase/*
```

这样，后面编译和运行过程才不会出错



4.3.2 读写HBase数据

3. 编写程序读取HBase数据

如果要让Spark读取HBase，就需要使用SparkContext提供的newAPIHadoopRDD这个API将表的内容以RDD的形式加载到Spark中。

SparkOperateHBase.py

```
#!/usr/bin/env python3
from pyspark import SparkConf, SparkContext
conf = SparkConf().setMaster("local").setAppName("ReadHBase")
sc = SparkContext(conf = conf)
host = 'localhost'
table = 'student'
conf = {"hbase.zookeeper.quorum": host, "hbase.mapreduce.inputtable": table}
keyConv = "org.apache.spark.examples.pythonconverters.ImmutableBytesWritableToStringConverter"
valueConv = "org.apache.spark.examples.pythonconverters.HBaseResultToStringConverter"
hbase_rdd =
sc.newAPIHadoopRDD("org.apache.hadoop.hbase.mapreduce.TableInputFormat", "org.apache.hadoop.h
base.io.ImmutableBytesWritable", "org.apache.hadoop.hbase.client.Result", keyConverter=keyConv, value
Converter=valueConv, conf=conf)
count = hbase_rdd.count()
hbase_rdd.cache()
output = hbase_rdd.collect()
for (k, v) in output:
    print (k, v)
```



4.3.2 读写HBase数据

执行该代码文件，命令如下：

```
$ cd /usr/local/spark/mycode/rdd
```

```
$ /usr/local/spark/bin/spark-submit SparkOperateHBase.py
```

执行后得到如下结果：

```
1 {"qualifier" : "age", "timestamp" : "1545728145163", "columnFamily" :  
  "info", "row" : "1", "type" : "Put", "value" : "23"}  
{"qualifier" : "gender", "timestamp" : "1545728114020", "columnFamily" :  
  "info", "row" : "1", "type" : "Put", "value" : "F"}  
{"qualifier" : "name", "timestamp" : "1545728100663", "columnFamily" :  
  "info", "row" : "1", "type" : "Put", "value" : "Xueqian"}  
2 {"qualifier" : "age", "timestamp" : "1545728184030", "columnFamily" :  
  "info", "row" : "2", "type" : "Put", "value" : "24"}  
{"qualifier" : "gender", "timestamp" : "1545728176815", "columnFamily" :  
  "info", "row" : "2", "type" : "Put", "value" : "M"}  
{"qualifier" : "name", "timestamp" : "1545728168727", "columnFamily" :  
  "info", "row" : "2", "type" : "Put", "value" : "Weiliang"}
```



4.3.2 读写HBase数据

4. 编写程序向HBase写入数据

下面编写应用程序把表中的两个学生信息插入到HBase的student表中

表 向student表中插入的新数据

id	info		
	name	gender	age
3	Rongcheng	M	26
4	Guanhua	M	27



4.3.2 读写HBase数据

在SparkWriteHBase.py文件中输入下面代码：

```
#!/usr/bin/env python3

from pyspark import SparkConf, SparkContext

conf = SparkConf().setMaster("local").setAppName("ReadHBase")
sc = SparkContext(conf = conf)
host = 'localhost'
table = 'student'
keyConv =
"org.apache.spark.examples.pythonconverters.StringToImmutableBytesWritableConverter"
valueConv = "org.apache.spark.examples.pythonconverters.StringListToPutConverter"
conf = {"hbase.zookeeper.quorum": host, "hbase.mapred.puttable":
table, "mapreduce.outputformat.class":
"org.apache.hadoop.hbase.mapreduce.TableOutputFormat", "mapreduce.job.output.key.class":
"org.apache.hadoop.hbase.io.ImmutableBytesWritable", "mapreduce.job.output.value.class":
"org.apache.hadoop.io.Writable"}
rawData =
['3,info,name,Rongcheng','3,info,gender,M','3,info,age,26','4,info,name,Guanhua','4,info,gender,M',
'4,info,age,27']
sc.parallelize(rawData).map(lambda x:
(x[0],x.split(','))).saveAsNewAPIHadoopDataset(conf=conf,keyConverter=keyConv,valueConverter=valueConv)
```



4.3.2 读写HBase数据

```
$ cd /usr/local/spark/mycode/rdd  
$ /usr/local/spark/bin/spark-submit SparkWriteHBase.py
```

切换到HBase Shell中，执行如下命令查看student表

```
hbase> scan 'student'
```

ROW	COLUMN+CELL
1	column=info:age, timestamp=1479640712163, value=23
1	column=info:gender, timestamp=1479640704522, value=F
1	column=info:name, timestamp=1479640696132, value=Xueqian
2	column=info:age, timestamp=1479640752474, value=24
2	column=info:gender, timestamp=1479640745276, value=M
2	column=info:name, timestamp=1479640732763, value=Weiliang
3	column=info:age, timestamp=1479643273142, value=\x00\x00\x00\x1A
3	column=info:gender, timestamp=1479643273142, value=M
3	column=info:name, timestamp=1479643273142, value=Rongcheng
4	column=info:age, timestamp=1479643273142, value=\x00\x00\x00\x1B
4	column=info:gender, timestamp=1479643273142, value=M
4	column=info:name, timestamp=1479643273142, value=Guanhua

4 row(s) in 0.3240 seconds



4.4 综合案例

4.4.1 案例1：求TOP值

4.4.2 案例2：文件排序

4.4.3 案例3：二次排序



4.4.1 案例1：求TOP值

任务描述：

orderid,userid,payment,productid

file1.txt

```
1,1768,50,155
2,1218, 600,211
3,2239,788,242
4,3101,28,599
5,4899,290,129
6,3110,54,1201
7,4436,259,877
8,2369,7890,27
```

file2.txt

```
100,4287,226,233
101,6562,489,124
102,1124,33,17
103,3267,159,179
104,4569,57,125
105,1438,37,116
```

求Top N个payment值



4.4.1 案例1：求TOP值

TopN.py

```
#!/usr/bin/env python3
```

```
from pyspark import SparkConf, SparkContext
```

```
conf = SparkConf().setMaster("local").setAppName("ReadHBase")
```

```
sc = SparkContext(conf = conf)
```

```
lines = sc.textFile("file:///usr/local/spark/mycode/rdd/file")
```

```
result1 = lines.filter(lambda line:(len(line.strip()) > 0) and (len(line.split(",")== 4))
```

```
result2 = result1.map(lambda x:x.split(",")[2])
```

```
result3 = result2.map(lambda x:(int(x),""))
```

```
result4 = result3.repartition(1)
```

```
result5 = result4.sortByKey(False)
```

```
result6 = result5.map(lambda x:x[0])
```

```
result7 = result6.take(5)
```

```
for a in result7:
```

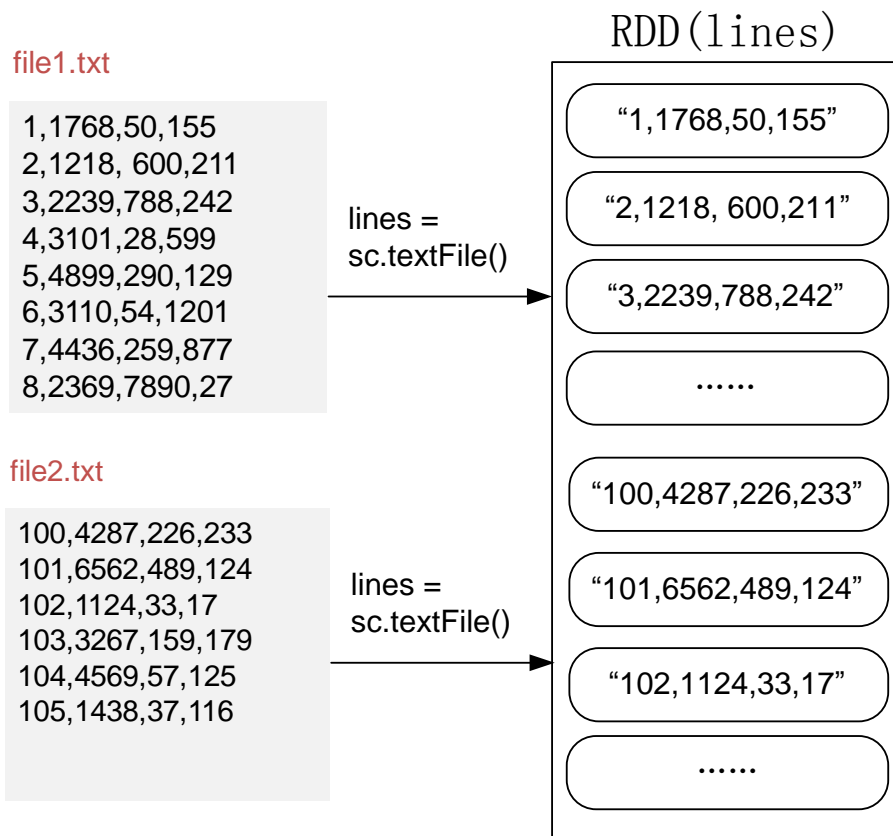
```
    print(a)
```



4.4.1 案例1：求TOP值

```
lines = sc.textFile("file:///usr/local/spark/mycode/rdd/file")
```

该语句从文件中读取数据生成RDD（名称为lines），执行后的效果如下：

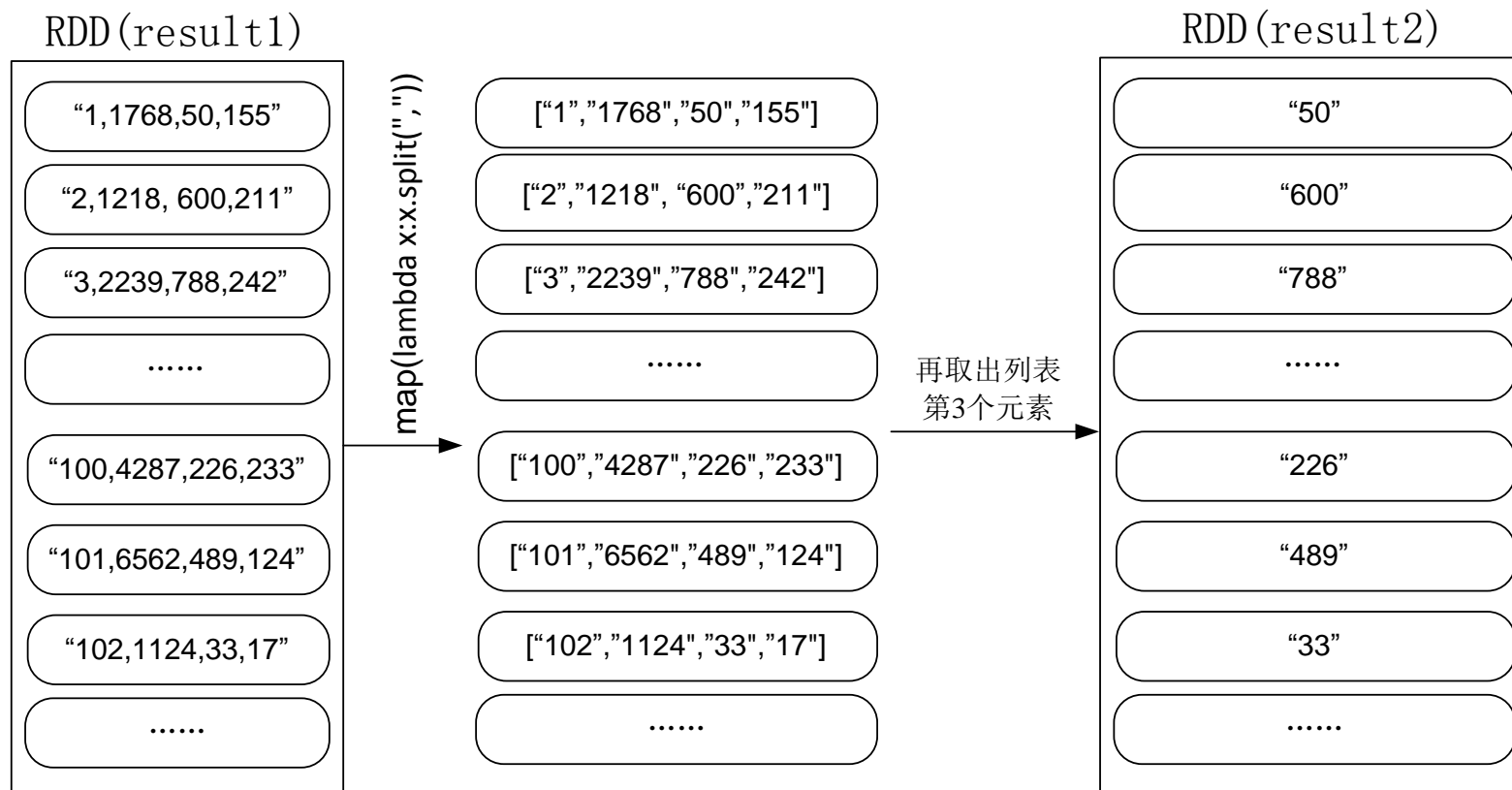




4.4.1 案例1：求TOP值

```
result1 = lines.filter(lambda line:(len(line.strip()) > 0) and (len(line.split(",")== 4))
result2 = result1.map(lambda x:x.split(",")[2])
```

该语句执行效果如下：

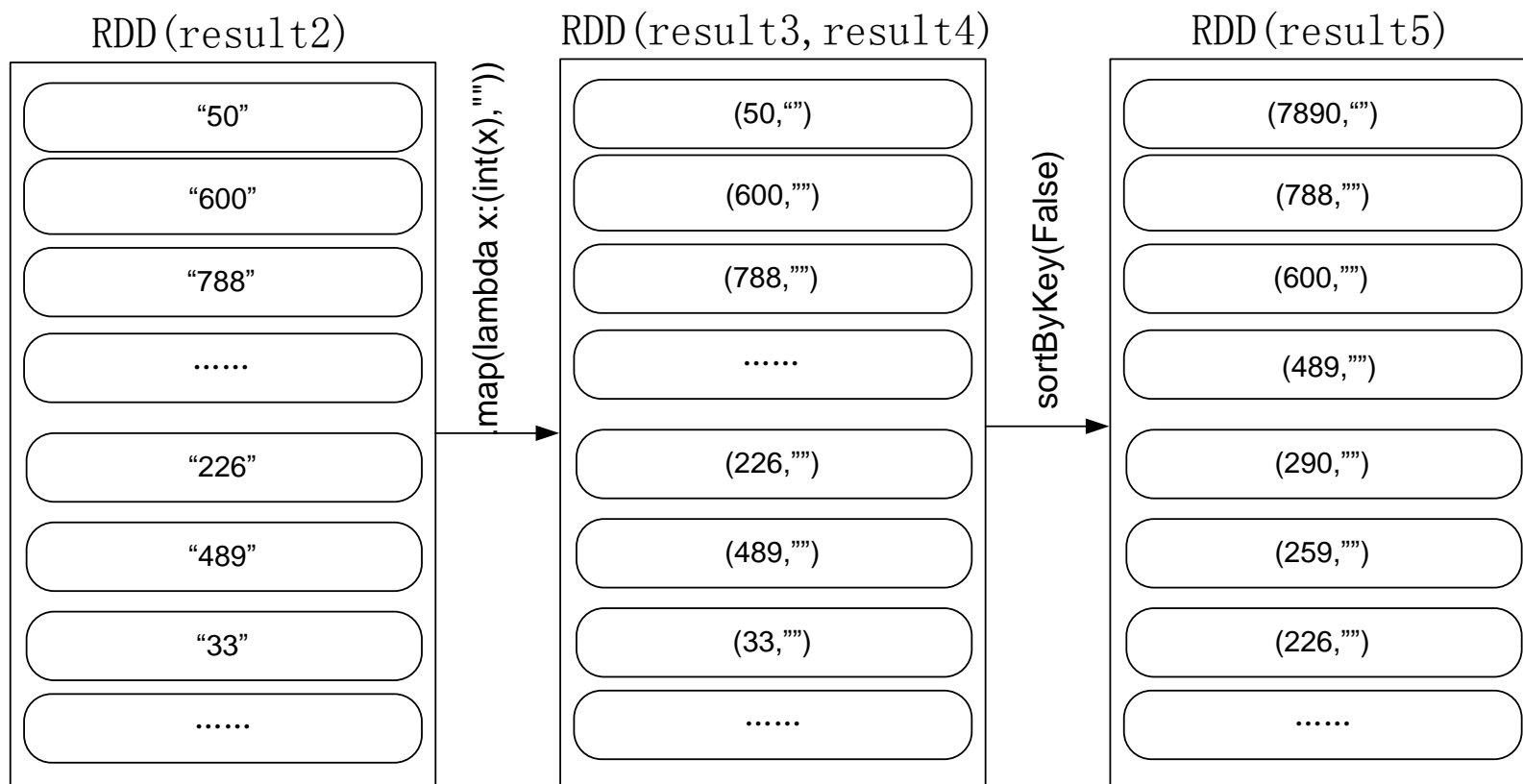




4.4.1 案例1：求TOP值

```
result3 = result2.map(lambda x:(int(x),""))
result4 = result3.repartition(1)
result5 = result4.sortByKey(False)
```

该语句执行效果如下：



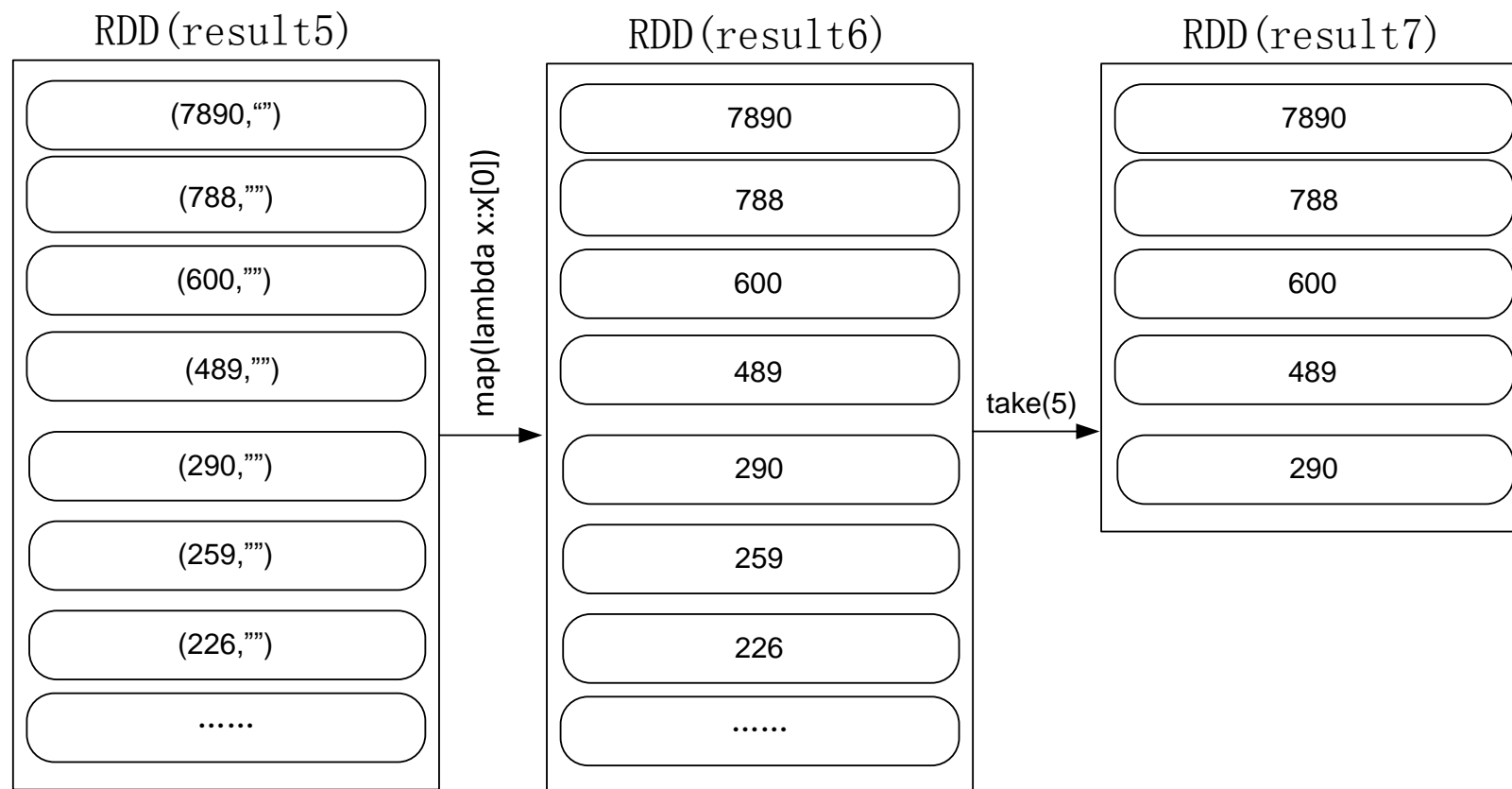


4.4.1 案例1：求TOP值

```
result6 = result5.map(lambda x:x[0])
```

```
result7 = result6.take(5)
```

该语句执行效果如下：





4.4.2 案例2：文件排序

任务描述：

有多个输入文件，每个文件中的每一行内容均为一个整数。要求读取所有文件中的整数，进行排序后，输出到一个新的文件中，输出的内容个数为每行两个整数，第一个整数为第二个整数的排序位次，第二个整数为原待排序的整数

输入文件

file1.txt

33

37

12

40

file2.txt

4

16

39

5

file3.txt

1

45

25

输出文件

1 1

2 4

3 5

4 12

5 16

6 25

7 33

8 37

9 39

10 40

11 45



4.4.2 案例2：文件排序

FileSort.py

```
#!/usr/bin/env python3

from pyspark import SparkConf, SparkContext

index = 0

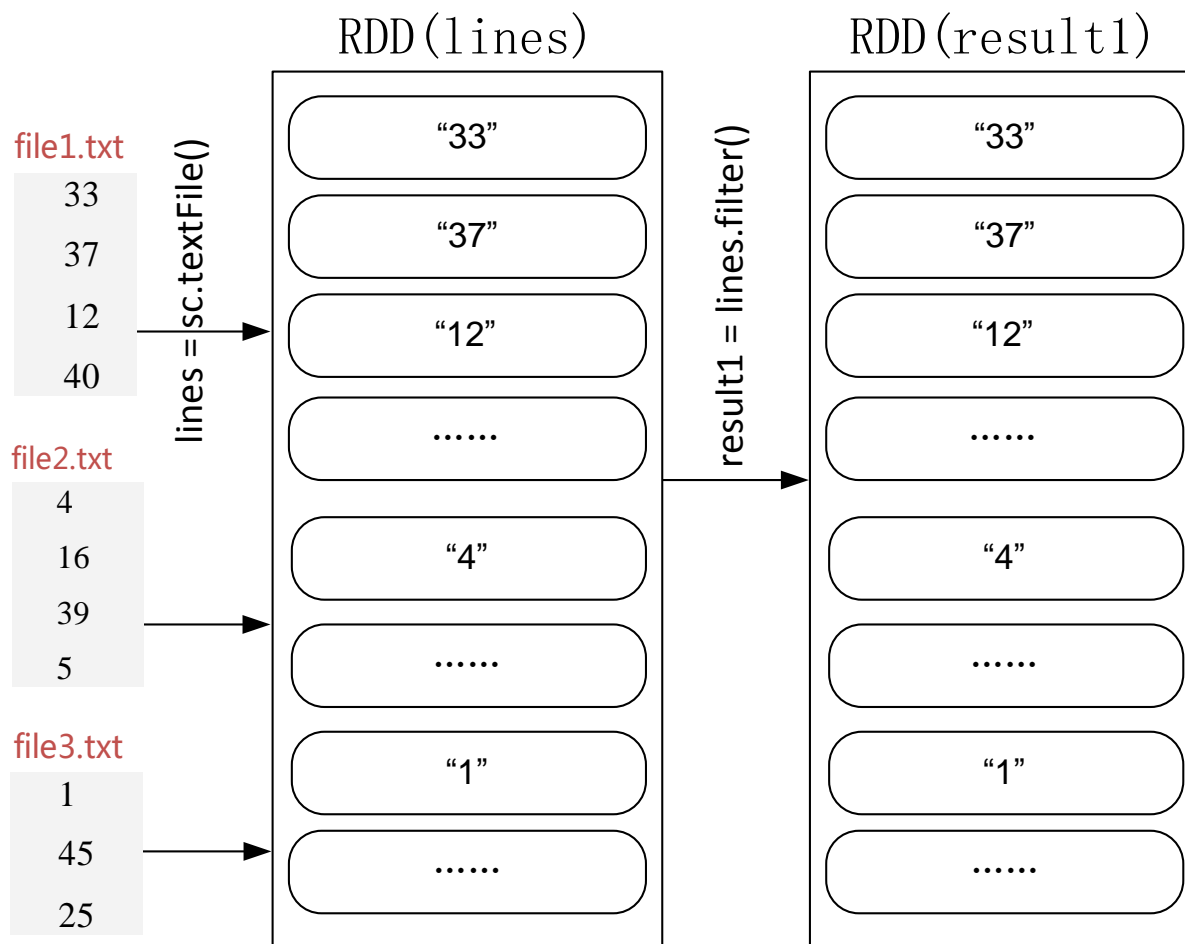
def getindex():
    global index
    index+=1
    return index

def main():
    conf = SparkConf().setMaster("local[1]").setAppName("FileSort")
    sc = SparkContext(conf = conf)
    lines = sc.textFile("file:///usr/local/spark/mycode/rdd/filesort/file*.txt")
    index = 0
    result1 = lines.filter(lambda line:(len(line.strip()) > 0))
    result2 = result1.map(lambda x:(int(x.strip()),""))
    result3 = result2.repartition(1)
    result4 = result3.sortByKey(True)
    result5 = result4.map(lambda x:x[0])
    result6 = result5.map(lambda x:(getindex(),x))
    result6.foreach(print)
    result6.saveAsTextFile("file:///usr/local/spark/mycode/rdd/filesort/sortresult")
if __name__ == '__main__':
    main()
```




4.4.2 案例2：文件排序

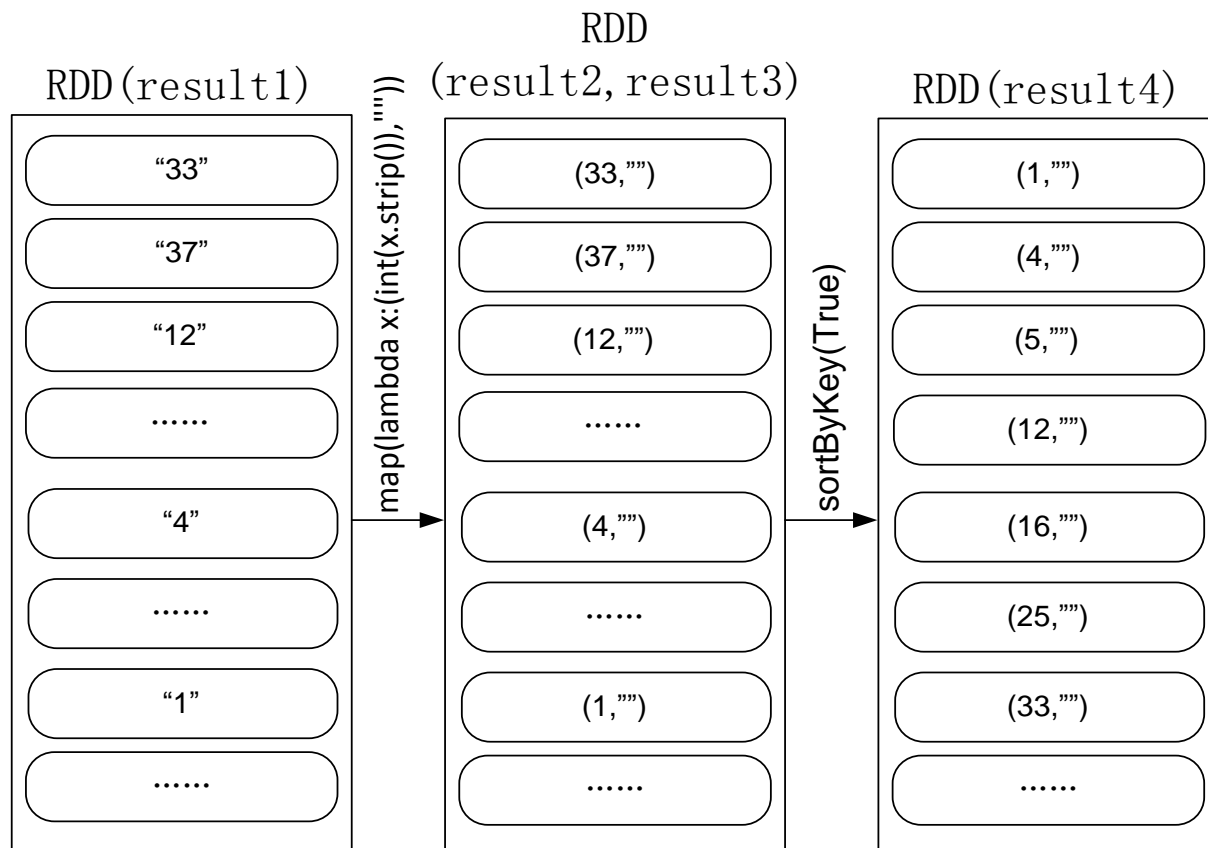
```
lines = sc.textFile("file:///usr/local/spark/mycode/rdd/filesort/file*.txt")
result1 = lines.filter(lambda line:(len(line.strip()) > 0))
```





4.4.2 案例2：文件排序

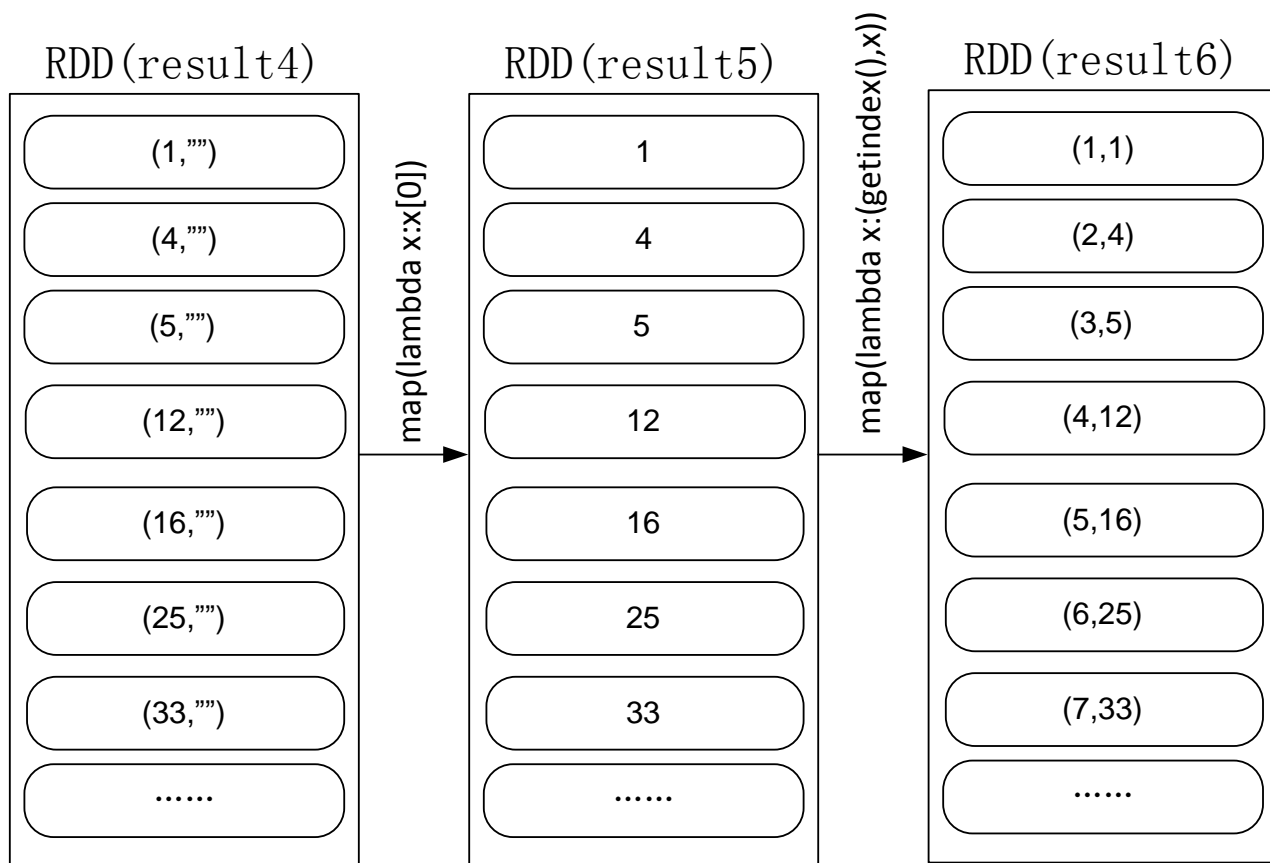
```
result2 = result1.map(lambda x:(int(x.strip()), ""))  
result3 = result2.repartition(1)  
result4 = result3.sortByKey(True)
```





4.4.2 案例2：文件排序

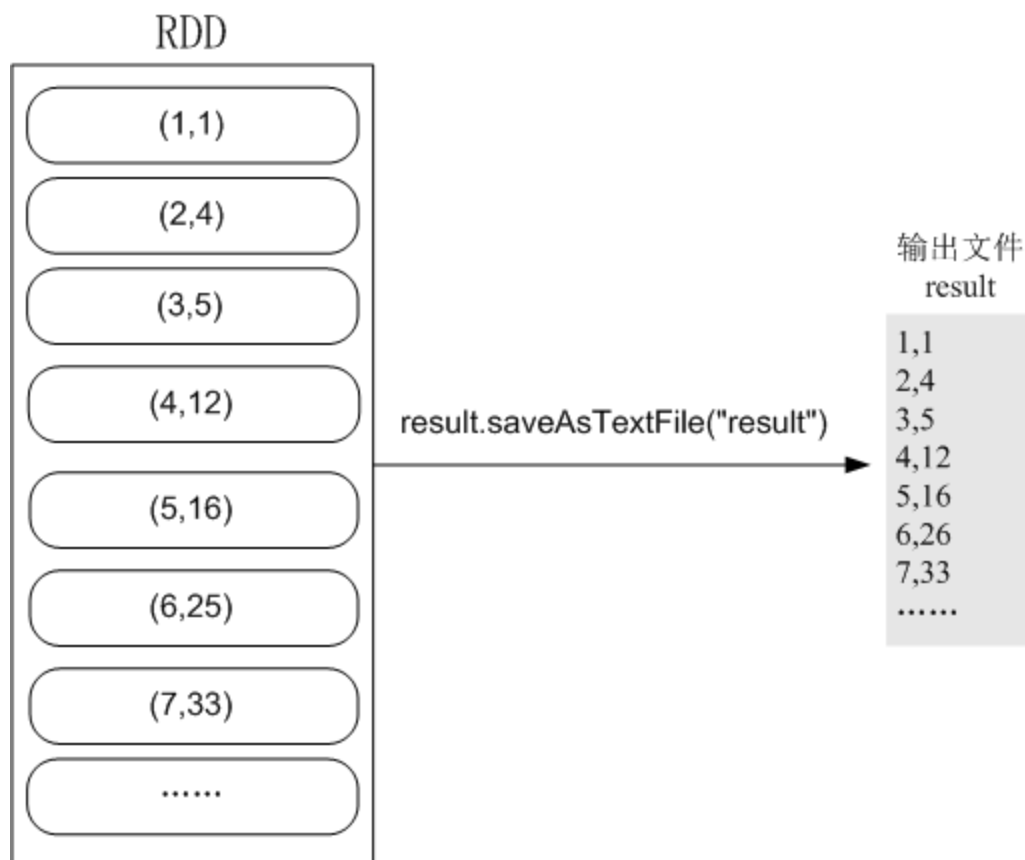
```
result5 = result4.map(lambda x:x[0])
result6 = result5.map(lambda x:(getIndex(),x))
```





4.4.2 案例2：文件排序

```
result6.saveAsTextFile("file:///usr/local/spark/mycode/rdd/filesort/sortresult")
```





4.4.3 案例3：二次排序

任务要求：

对于一个给定的文件（数据如file1.txt所示），请对数据进行排序，首先根据第1列数据降序排序，如果第1列数据相等，则根据第2列数据降序排序

输入文件file1.txt

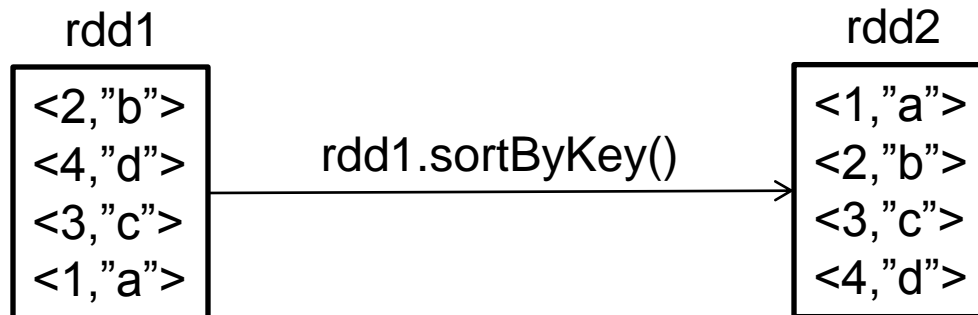
5	3
1	6
4	9
8	3
4	7
5	6
3	2

输出结果

8	3
5	6
5	3
4	9
4	7
3	2
1	6



4.4.3 案例3：二次排序



用于排序的key必须是可比较的对象

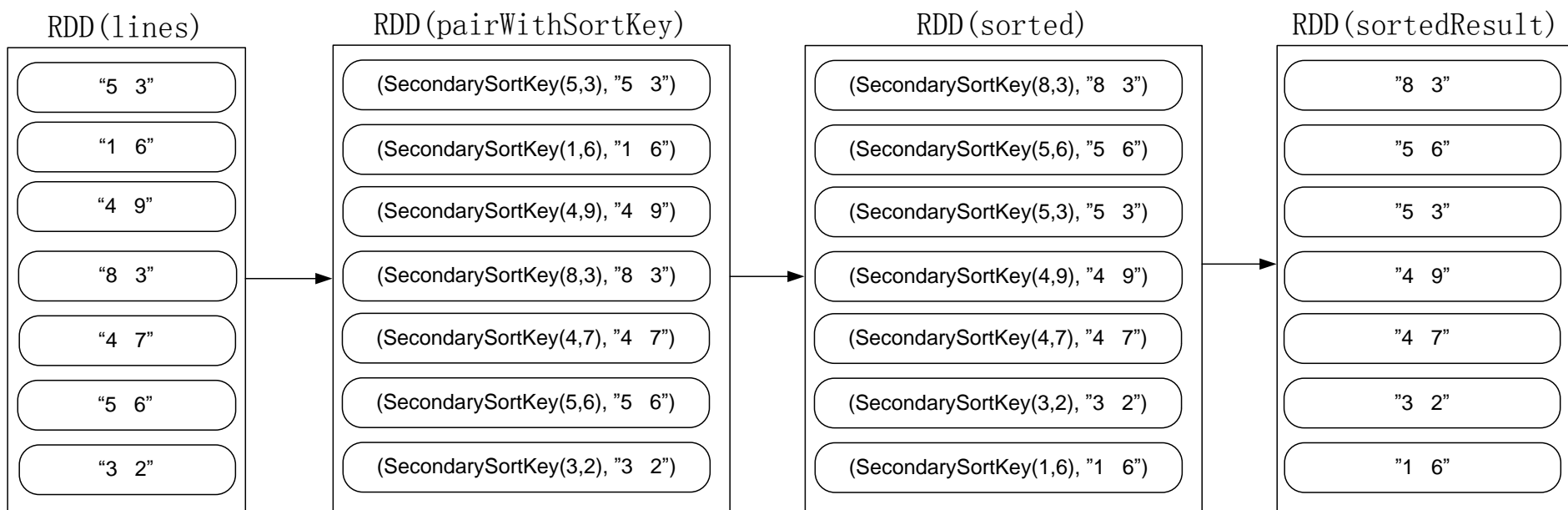
"5	3"
"1	6"
"4	9"
"8	3"
"4	7"
"5	6"
"3	2"

如何比较？



4.4.3 案例3：二次排序

解题思路：





4.4.3 案例3：二次排序

二次排序，具体的实现步骤如下

01

第一步：定义一个类SecondarySortKey，实现自定义的用于排序的key

02

第二步：将要进行二次排序的文件加载进来生成(key,value)类型的RDD

03

第三步：使用sortByKey()基于自定义的key进行二次排序

04

第四步：去除掉排序的key，只保留排序的结果



4.4.3 案例3：二次排序

SecondarySortKey.py代码如下：

```
#!/usr/bin/env python3

from operator import gt
from pyspark import SparkContext, SparkConf

class SecondarySortKey():
    def __init__(self, k):
        self.column1 = k[0]
        self.column2 = k[1]

    def __gt__(self, other):
        if other.column1 == self.column1:
            return gt(self.column2, other.column2)
        else:
            return gt(self.column1, other.column1)
```

剩余代码见下一页



4.4.3 案例3：二次排序

SecondarySortApp.py代码如下：

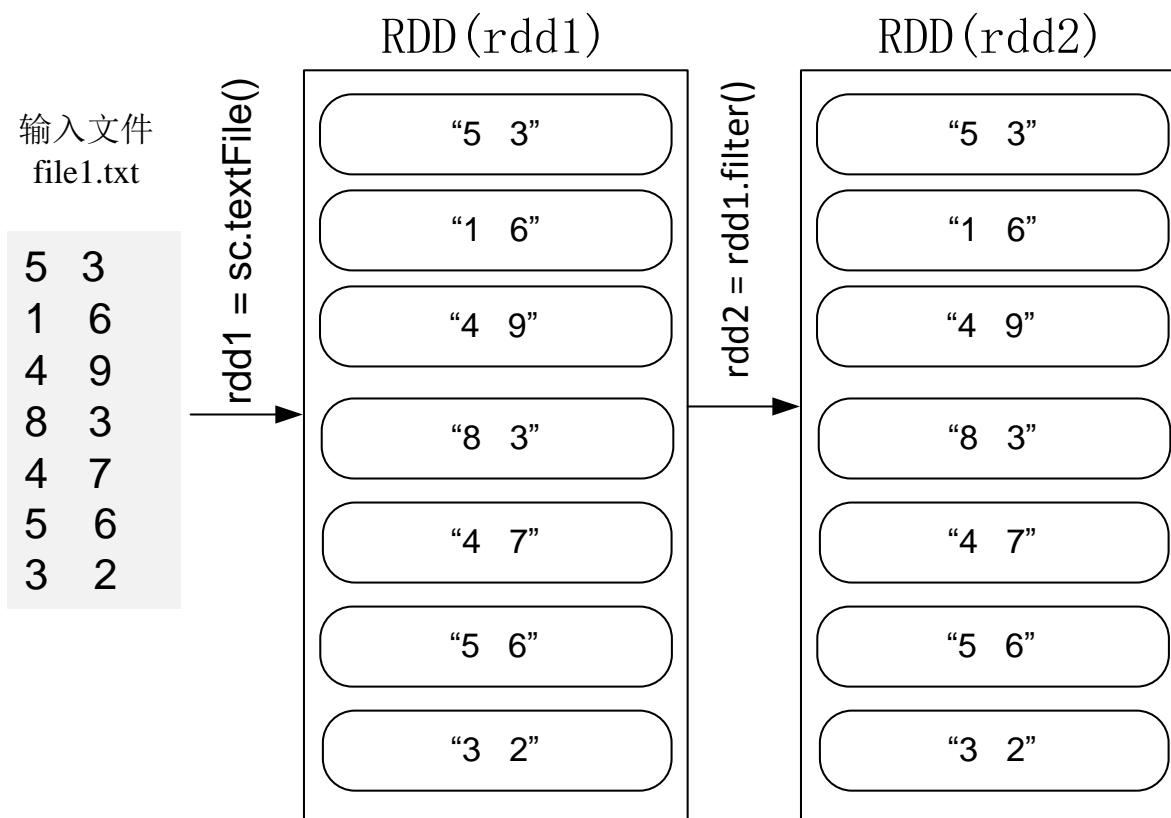
```
def main():
    conf = SparkConf().setAppName('spark_sort').setMaster('local[1]')
    sc = SparkContext(conf=conf)
    file="file:///usr/local/spark/mycode/rdd/secondarysort/file4.txt"
    rdd1 = sc.textFile(file)
    rdd2 = rdd1.filter(lambda x:(len(x.strip())) > 0))
    rdd3 = rdd2.map(lambda x:((int(x.split(" ")[0]),int(x.split(" ")[1])),x))
    rdd4 = rdd3.map(lambda x: (SecondarySortKey(x[0]),x[1]))
    rdd5 = rdd4.sortByKey(False)
    rdd6 = rdd5.map(lambda x:x[1])
    rdd6.foreach(print)

if __name__ == '__main__':
    main()
```



4.4.3 案例3：二次排序

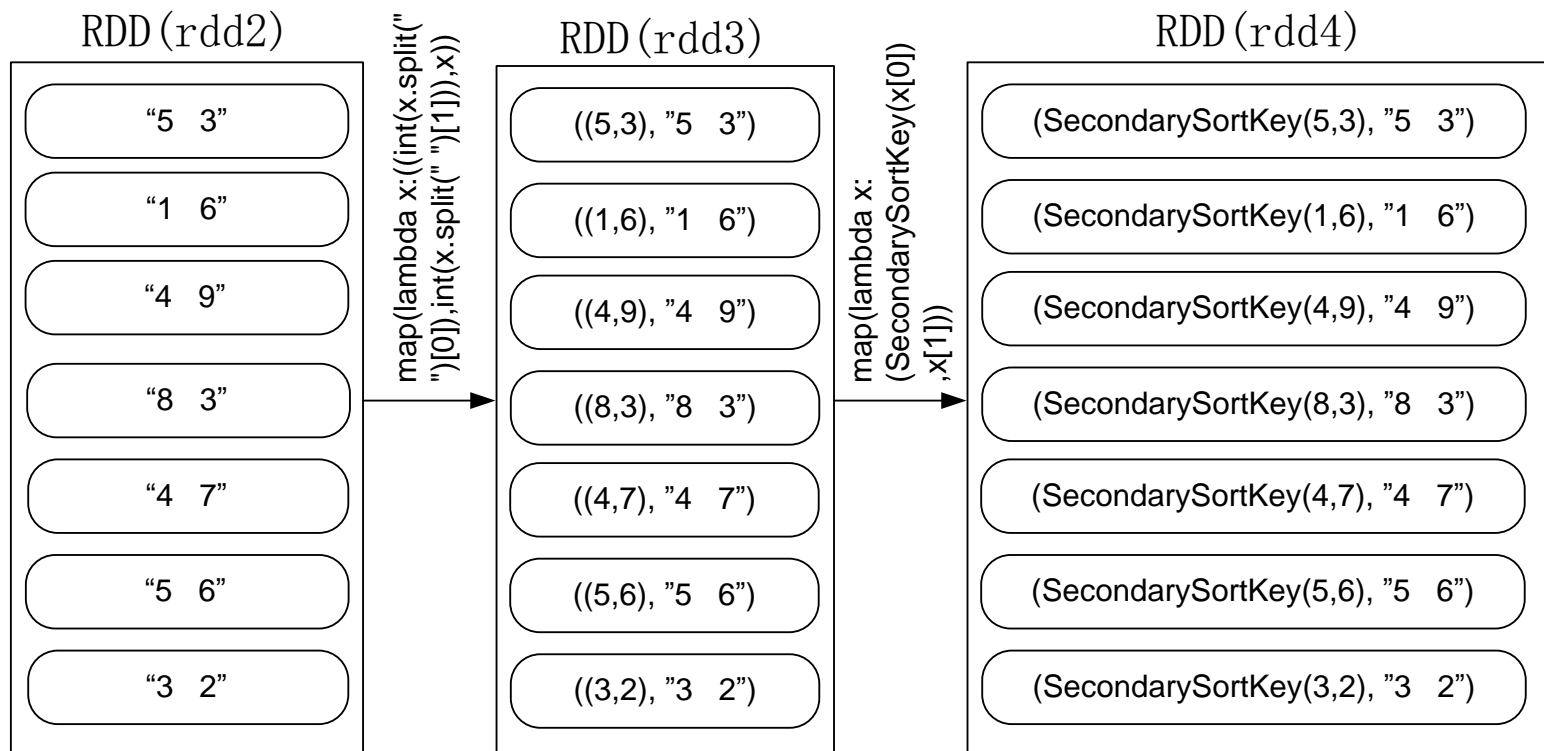
```
rdd1 = sc.textFile(file)
rdd2 = rdd1.filter(lambda x:(len(x.strip()) > 0))
```





4.4.3 案例3：二次排序

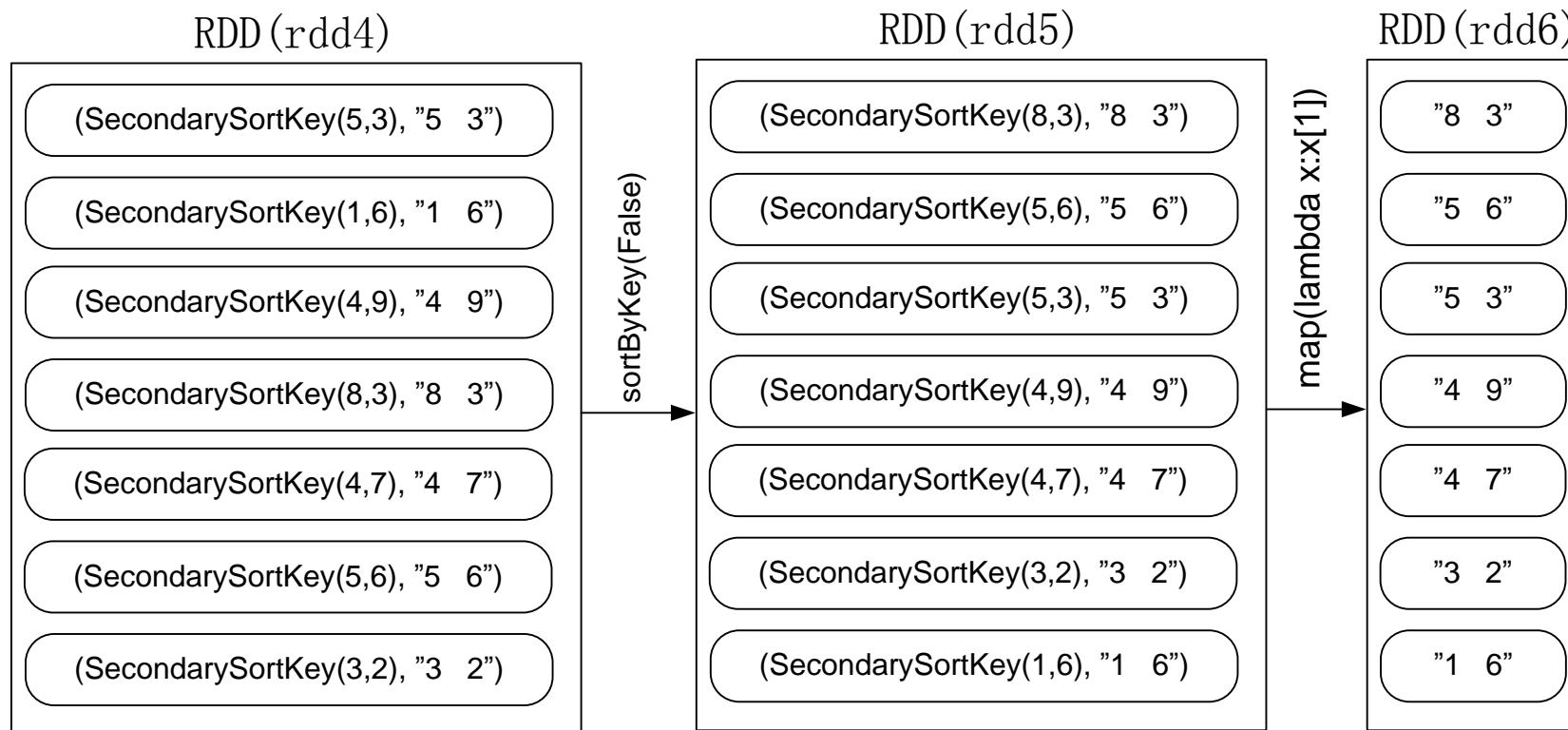
```
rdd3 = rdd2.map(lambda x:((int(x.split(" ")[0]),int(x.split(" ")[1])),x))
rdd4 = rdd3.map(lambda x: (SecondarySortKey(x[0]),x[1]))
```





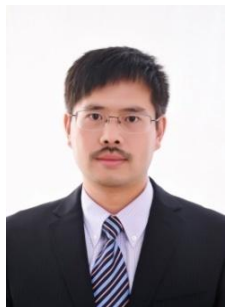
4.4.3 案例3：二次排序

```
rdd5 = rdd4.sortByKey(False)
rdd6 = rdd5.map(lambda x:x[1])
```





附录A：主讲教师林子雨简介



主讲教师：林子雨

单位：厦门大学计算机科学系

E-mail: ziyulin@xmu.edu.cn

个人网页: <http://dblab.xmu.edu.cn/post/linziyu>

数据库实验室网站: <http://dblab.xmu.edu.cn>



扫一扫访问个人主页

林子雨，男，1978年出生，博士（毕业于北京大学），现为厦门大学计算机科学系助理教授（讲师），曾任厦门大学信息科学与技术学院院长助理、晋江市发展和改革委员会副局长。中国计算机学会数据库专业委员会委员，中国计算机学会信息系统专业委员会委员。国内高校首个“数字教师”提出者和建设者，厦门大学数据库实验室负责人，厦门大学云计算与大数据研究中心主要建设者和骨干成员，2013年度和2017年度厦门大学教学类奖教金获得者，荣获2017年福建省精品在线开放课程、2018年厦门大学高等教育成果特等奖、2018年福建省高等教育教学成果二等奖、2018年国家精品在线开放课程。主要研究方向为数据库、数据仓库、数据挖掘、大数据、云计算和物联网，并以第一作者身份在《软件学报》《计算机学报》和《计算机研究与发展》等国家重点期刊以及国际学术会议上发表多篇学术论文。作为项目负责人主持的科研项目包括1项国家自然科学基金青年基金项目(No.61303004)、1项福建省自然科学基金项目(No.2013J05099)和1项中央高校基本科研业务费项目(No.2011121049)，主持的教改课题包括1项2016年福建省教改课题和1项2016年教育部产学协作育人项目，同时，作为课题负责人完成了国家发改委城市信息化重大课题、国家物联网重大应用示范工程区域试点泉州市工作方案、2015泉州市互联网经济调研等课题。中国高校首个“数字教师”提出者和建设者，2009年至今，“数字教师”大平台累计向网络免费发布超过500万字高价值的研究和教学资料，累计网络访问量超过500万次。打造了中国高校大数据教学知名品牌，编著出版了中国高校第一本系统介绍大数据知识的专业教材《大数据技术原理与应用》，并成为京东、当当网等网店畅销书籍；建设了国内高校首个大数据课程公共服务平台，为教师教学和学生学习大数据课程提供全方位、一站式服务，年访问量超过200万次。





附录C：《大数据技术原理与应用》教材

《大数据技术原理与应用——概念、存储、处理、分析与应用（第2版）》，由厦门大学计算机科学系林子雨博士编著，是国内高校第一本系统介绍大数据知识的专业教材。人民邮电出版社 ISBN:978-7-115-44330-4 定价：49.80元



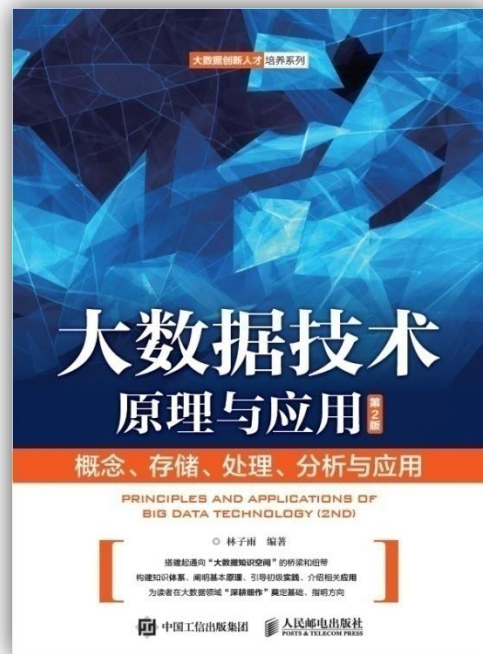
扫一扫访问教材官网

全书共有15章，系统地论述了大数据的基本概念、大数据处理架构Hadoop、分布式文件系统HDFS、分布式数据库HBase、NoSQL数据库、云数据库、分布式并行编程模型MapReduce、Spark、流计算、图计算、数据可视化以及大数据在互联网、生物医学和物流等各个领域的应用。在Hadoop、HDFS、HBase和MapReduce等重要章节，安排了入门级的实践操作，让读者更好地学习和掌握大数据关键技术。

本书可以作为高等院校计算机专业、信息管理等相关专业的大数据课程教材，也可供相关技术人员参考、学习、培训之用。

欢迎访问《大数据技术原理与应用——概念、存储、处理、分析与应用》教材官方网站：

<http://dbllab.xmu.edu.cn/post/bigdata>





附录D：《大数据基础编程、实验和案例教程》

本书是与《大数据技术原理与应用（第2版）》教材配套的唯一指定实验指导书

大数据教材



1+1黄金组合
厦门大学林子雨编著

配套实验指导书



- 步步引导，循序渐进，详尽的安装指南为顺利搭建大数据实验环境铺平道路
- 深入浅出，去粗取精，丰富的代码实例帮助快速掌握大数据基础编程方法
- 精心设计，巧妙融合，五套大数据实验题目促进理论与编程知识的消化和吸收
- 结合理论，联系实际，大数据课程综合实验案例精彩呈现大数据分析全流程

清华大学出版社 ISBN:978-7-302-47209-4 定价：59元

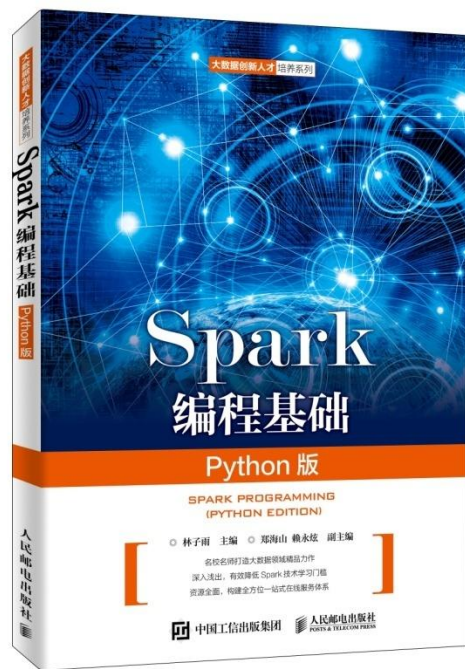


附录E：《Spark编程基础（Python版）》

林子雨，郑海山，赖永炫 编著 《**Spark编程基础（Python版）**》

教材官网：<http://dblab.xmu.edu.cn/post/spark-python/>

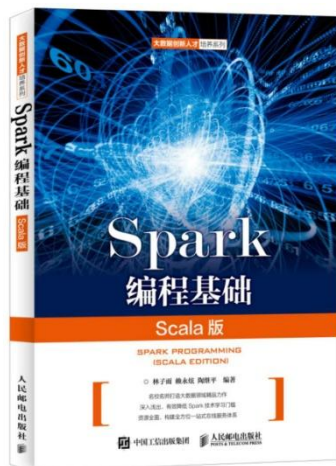
ISBN:978-7-115-52439-3 人民邮电出版社



本书以Python作为开发Spark应用程序的编程语言，系统介绍了Spark编程的基础知识。全书共8章，内容包括大数据技术概述、Spark的设计与运行原理、Spark环境搭建和使用方法、RDD编程、Spark SQL、Spark Streaming、Structured Streaming、Spark MLlib等。本书每个章节都安排了入门级的编程实践操作，以便读者更好地学习和掌握Spark编程方法。本书官网免费提供了全套的在线教学资源，包括讲义PPT、习题、源代码、软件、数据集、授课视频、上机实验指南等。



附录F：《Spark编程基础（Scala版）》



《Spark编程基础（Scala版）》

厦门大学 林子雨，赖永炫，陶继平 编著

披荆斩棘，在大数据丛林中开辟学习捷径
填沟削坎，为快速学习Spark技术铺平道路
深入浅出，有效降低Spark技术学习门槛
资源全面，构建全方位一站式在线服务体系

人民邮电出版社出版发行，ISBN:978-7-115-48816-9
教材官网：<http://dblab.xmu.edu.cn/post/spark/>



本书以Scala作为开发Spark应用程序的编程语言，系统介绍了Spark编程的基础知识。全书共8章，内容包括大数据技术概述、Scala语言基础、Spark的设计与运行原理、Spark环境搭建和使用方法、RDD编程、Spark SQL、Spark Streaming、Spark MLlib等。本书每个章节都安排了入门级的编程实践操作，以便读者更好地学习和掌握Spark编程方法。本书官网免费提供了全套的在线教学资源，包括讲义PPT、习题、源代码、软件、数据集、授课视频、上机实验指南等。



附录G：高校大数据课程公共服务平台



高校大数据课程

公 共 服 务 平 台

<http://dblab.xmu.edu.cn/post/bigdata-teaching-platform/>



扫一扫访问平台主页



扫一扫观看3分钟FLASH动画宣传片

The background is a solid blue color. It features several faint, light-blue silhouettes of people. In the top left, there is a group of three people holding hands. In the top center, there is a group of seven people holding hands. On the right side, there is a large silhouette of a person standing with their hand on their chin. In the bottom left, there is a silhouette of a person's head and shoulders.

Thank You!

Department of Computer Science, Xiamen University, 2020